İşe kesinliklerle başlayan, şüphelerle bitirir. Şüphelerle başlamaya razı olan, kesinliklerle bitirir.

Francis Bacon

Bölüm

1

Giriş

P

rofesyonel Programlama Teknikleri.NET, kaliteli yazılımlar elde etmek amacıyla kullanılabilecek mühendislik prensip ve metodolojileri, kavramsal ve teorik boyutlu programlama bilgileri, pratikte uygulanan teknik ve yaklaşımlar konularını Microsoft.NET platformunu temel alarak aktaran başvuru ve kaynak kitabıdır.

Kitabın en önemli amacı, Microsoft.NET platformu ve teknolojileri ile birlikte profesyonel programlama teknikleri ve yaklaşımlarını bütün olarak aktarabilmek, teorik konuların pratikte uygulanma modelleri hakkında bilgi vermek, okuyucunun gelecekteki bilgisini artırabilmesi amacıyla gerekli yolu çizmek ve böylece okuyucunun büyük .NET resmini görebilmesini sağlamaktır.

Kitap Organizasyonu

Kitapta aktarılan bilgiler, bölümler ve bölümler içerisinde bulunan konu başlıkları şeklinde hiyerarşik olarak organize edilmiştir. Her bölüm içerisinde sunulan bilgiler bir bütün olarak aktarılmış, gerektiği durumlarda kitapta bulunan başka bölüm ve konu başlıklarına referans verilmiştir.

Bölüm sıralaması, bölümlerin birbirlerine bağımlılıkları baz alınarak yapılmıştır. Bir başka anlatımla, bir bölüm içerisinde bulunan konuların okuyucu tarafından anlaşılabilmesi için gerekli bilgilerin önceki bölümlerde sağlanmış olması temel alınmıştır.

Böylece, okuyucunun kitabı başından sonuna kadar okuması, örnek kod parçalarını denemesi ve senaryoları incelemesi ile .NET konusunda bilgi sahibi olması sağlanmıştır.

Anlatım Modeli

Kitap anlatım modeli belirli bir sistematik kullanılarak organize edilmiştir. Kitapta bulunan bölümlerde öncelikle kavramsal ve teorik düzeyde bilgiler okuyucuya sunulmuştur. Böylece okuyucunun .NET tarafından sunulan çözümlerden bağımsız olarak aktarılan bilginin ne anlam ifade ettiği, hangi problemlere çözüm sağladığı hakkında bilgi sahibi olması sağlanmıştır.

Kavramsal ve teorik düzeyde aktarılan bilgilerin devamında, Microsoft.NET teknolojisinin sunduğu çözüm ve mimari hakkında genel bilgiler verilmiştir. Verilen bilgiler, .NET tarafından sunulan çözüm ve yaklaşımların detayları ve sınırları konularında yoğunlaşmıştır. .NET sınıf kütüphanesi içerisinde bulunan sınıfların kullanımları, üye bilgileri, örnek kod parçaları ile aktarılan bilgiler desteklenmiştir. Örnek kod parçaları C# ve VB.NET programlama dillerinde birlikte sunulmuş, böylece okuyucunun her iki programlama dili ile yapılabilecek gerçekleştirim detaylarını görebilmesi sağlanmıştır.

Gerekli görüldüğü durumlarda bölüm örnek senaryo ile tamamlanmıştır. Örnek senaryo, okuyucunun gerçek hayatta karşılaşabileceği problemlere aktarılan bilgiler ışığında ne şekilde çözüm bulabileceği, probleme yaklaşım tarzı ve gerçekleştirim detaylarından oluşmaktadır.

Kullanılan Simge ve Stiller

Okuyucuyu uyarmak, bir başka kaynağa referans vermek veya önerilerde bulunmak amacıyla farklı simge ve stil kullanımı benimsenmiştir.

Aşağıdaki tabloda kitapta kullanılan simgeler ve açıklamaları gösterilmiştir.

| Simge | Açıklama |
| --- | --- |
|  | İpucu vermek amacıyla kullanılmıştır. İpuçları, konunun dışında olan ancak okuyucunun bilgi sahibi olmasında fayda görülen, konu ile ilgili ancak anlatım kapsamında ele alınmayan bilgileri ve referansları içermektedir. |
|  | Detay bilgi vermek amacıyla kullanılmıştır. Detay bilgi, konu anlatımında genel düzeyde bulunan bilgilerin gerektiği durumlarda okuyucuya daha detaylı olarak aktarım amacıyla kullanılmıştır. |
|  | Uyarı vermek amacıyla kullanılmıştır. Uyarı, konu hakkında okuyucunun yanlış anlamasını engellemek veya yapılan sık hatalar hakkında okuyucuya bilgi vermek amacıyla kullanılmıştır. |
|  | Okuyucunun konu hakkında analiz yapabilmesini sağlamak ve devamında aktarılan bilgileri daha iyi kavramasını sağlayabilmek amacıyla sorulan soruyu belirtmek amacıyla kullanılmıştır. |
|  | Okuyucunun bir sonraki bölüm veya konu başlığına geçmeden önce, ilgili bölüm içerisinde aktarılan bilgiler ışığında pratik çalışmaları yapması gerektiğini belirtmek için kullanılmıştır. |
|  | Listelenen kaynak kodlar veya çeşitli araçların kitap ile dağıtılan DVD içerisindeki yerleşimini belirtmek amacıyla kullanılmıştır. |
|  | Sınıf üye bilgilerinin listelendiği tablolarda ilgili sınıf üyesinin metot olduğunu belirtmek amacıyla kullanılmıştır. |
|  | Sınıf üye bilgilerinin listelendiği tablolarda ilgili sınıf üyesinin özellik olduğunu belirtmek amacıyla kullanılmıştır. |

Tablo 1, Kitapta Kullanılan Simgeler ve Açıklamaları

Okuyucunun görsel fark yardımıyla daha kolay okuyabilmesini veya özellikle vurgulanmak istenen kelime ve cümleleri belirtmek amacıyla kitapta farklı stiller kullanılmıştır.

| Stil | Açıklama |
| --- | --- |
| Kavram ve Tanım, İngilizce Kelime, İngilizce Karşılık, Senaryo Açıklaması | Stil farklı amaçlar için kullanılmıştır. Kullanım amacı bir kelime veya cümleye vurgu yapmaktan senaryo açıklamasına kadar değişiklik göstermektedir. |
| Kaynak Kod | C#, VB.NET, Delphi.NET ve SQL kaynak kodlarını listelemek amacıyla kullanılan stildir. |
| Çıktı | Aktarılan kod parçası veya komut çıktısını listelemek amacıyla kullanılmıştır. |

Tablo 2, Kitapta Kullanılan Temel Stiller ve Açıklamaları

DVD

Kitap ile birlikte dağıtılan DVD okuyucuya zengin içerik sağlamaktadır. DVD içerisinde kitapta listelenen kaynak kodların tamamı, örnek uygulamalar ile programlama araçları ve yardımcı yazılımlar bulunmaktadır. Kitapta referansı gösterilmeyen bir çok araç da DVD ile dağıtılmaktadır.

Dağıtılan yazılımların kanuni hakları tamamen üreticisine ait olup kitap yayıncısı tarafından yazılımların sadece dağıtımı yapılmaktadır.

Silgi kullanmadan resim çizme sanatına hayat denilmektedir.

John Christian

Bölüm

2

Temel Kavram ve Tanımlar

M

icrosoft tarafından 2000 yılında yol haritası çizilen ve 2002 yılında ilk sürümü ile kullanıma sunulan .NET, hizmet odaklı mimari (ing. Service Oriented Architecture - SOA), web uygulamaları, XML web hizmetleri (ing. XML Web Services) ve smart-istemci (ing. Smart Client) uygulama yazılımları gibi misyon-kritik (ing. Mission Critical), ölçeklenebilir (ing. Scalable), güvenli (ing. Secure) ve güvenilir (ing. Reliable) yazılım çözümleri için platform oluşturmakta ve temel yazılım mimarisi sağlamaktadır.

Bu bölümde, önümüzdeki bir kaç on yıl için Microsoft'un yaptığı en önemli yatırımlardan birisi olarak değerlendirilen .NET teknolojisi hakkında kavramsal düzeyde temel ve teorik bilgiler verilecektir.

.NET temelleri konusunda bilgi sahibi olduğunu düşünen okuyucular bu bölümü atlayabilirler. Buna rağmen, özellikle verilen teorik bilgilerin kitabın diğer bölümlerinde kullanılması, .NET teknolojisinin geneli itibariyle daha sağlıklı anlaşılabilmesi ve değerlendirilebilmesi için tüm okuyucuların bu bölümü okuması önerilmektedir.

Yazılım Temelleri

Yazılım (ing. Software) ve Yazılım Mühendisliği (ing. Software Engineering) kavramsal tanımlarını aşağıdaki şekliyle yapmak mümkündür.

Yazılım, komut düzeni önceden belirlenmiş bir donanım tarafından çalıştırılan komutlar dizesidir.

Yazılım Mühendisliği, kaliteli yazılım elde etmek amacıyla mühendislik prensiplerinin kullanımıdır.

Yazılım mühendisliği prensip ve disiplinleri bu kitabın konusu dışında olmakla birlikte, bir yazılım projesinde bulunan tüm ekip üyeleri tarafından yazılım mühendisliği disiplin ve kurallarının anlaşılması ve pratikte uygulanabilir düzeyde bilinmesi nihai yazılım başarısı için büyük önem taşımaktadır.

Donanımsal Mimari

Bilgisayar kavramı değerlendirildiğinde aşağıdaki kısa tanım yapılabilir.

Bilgisayar, girdi bilgileri işleyen, belirlenmiş kurallara göre işleme yaptıktan sonra bu bilgilerden çıktı bilgiler oluşturan ve bu bilgileri gerektiğinde ulaşılabilecek şekilde saklayabilen cihazdır.

Yukarıda yapılan tanımda anahtar kelimeler girdi bilgi, işleme ve çıktı bilgi olarak özetlenebilir.

Girdi bilgiler, kullanıcı tarafından klavye, fare gibi aygıtlar kullanılarak iletilen bilgiler olabileceği gibi bilgisayara farklı arabirimler (ing. Interfaces) (USB, Seri Port, Ağ Bağdaştırıcısı) yardımıyla iletilen bilgiler de olabilir.

Merkezi İşlemci Birimi (ing. Central Processing Unit - CPU) olarak tanımlanan işleme birimi yardımıyla girdi bilgilerin dönüşümü, yorumlanması gibi işleme olarak tanımlanabilecek işlemler yapılır.

Çıktı bilgilerin, işlenmiş olan bilgilerin, dış dünyaya farklı arabirimler (Monitör, yazıcı, ağ bağdaştırıcısı, vb.) yardımıyla aktarımı ile bir bilgisayarın temel işlevi tamamlanmış olur.

Merkezi İşleme Birimi

Bilgisayar tarafından işleme yapılabilmesi için gerek duyulan komut dizelerinin belirli kurallara uyularak hazırlanmış olması gerekmektedir. Makine Kodu (ing. Machine Code) olarak tanımlanabilecek bu dizelerin en küçük birimi İkil (Binary Digit - Bit) olarak ifade edilmektedir. Bir ikil, (0) veya (1) değerini alabilecek en küçük depolanabilir değer olarak tanımlanmaktadır. Bir başka anlatımla ikil, iki durumu ifade edebilen (örneğin açık – kapalı, var – yok, artı – eksi, vb.) en küçük değerler kümesinden oluşmaktadır.

İşletim Sistemleri

İşletim sistemleri (ing. Operating Systems) bilgisayar üzerinde çalışan en önemli yazılımlardır. İşletim sisteminin temel görevleri, uygulama yazılımlarının çalıştırılması, girdi ve çıktı bilgilerin yönetimi, dosya, klasör ve disk bilgilerinin saklanması gibi alt seviye (ing. Low level) olarak nitelenen işlemlerden oluşmaktadır.

Şekil 1, İşletim Sistemi

İşletim sistemleri, uygulama yazılımlarının çalışması için platform sağlamaktadırlar. Buradan çıkarılması gereken önemli bir sonuç, bir uygulama yazılımının geliştirilirken hangi işletim sistemi üzerinde çalışacağının önceden belirlenmesinin gerektiğidir.

Platform Bağımsız olarak nitelenen bazı uygulama yazılım türleri (örneğin Java), sanal bir donanım üzerinde (örneğin Java Virtual Machine - JVM) çalışmaktadır. Buna rağmen, işletim sistemi ve uygulama yazılımı arasında çalışan sanal platform her işletim sistemine göre farklı şekilde hazırlanmaktadır. Örneğin Windows işletim sistemi üzerinde bir Java uygulamasını çalıştırabilmek için Windows için hazırlanmış olan JVM yazılımı sisteme yüklenmiş olmalıdır. Aynı uygulama yazılımı Sun Solaris işletim sistemi üzerinde herhangi bir yeniden derleme yapmadan çalışabilmesine rağmen, bu durumda ilgili donanım üzerinde Sun Solaris için hazırlanmış olan JVM yazılımının yüklü olması gerekmektedir. Her iki durumda da uygulama yazılımı aynı kalsa da Windows için JVM ve Sun Solaris için JVM farklıdır.

Derleme İşlemi

Derleme (ing. Compile), kullanıcı okunabilir kodun (kaynak kodun) makine okunabilir koda çevrimi olarak tanımlanabilir. Kavramın dönüştürme (ing. Conversion) olarak ifade edilmemesindeki temel sebep, işlem sırasında dönüştürme dışında bazı adımların da gerçekleşiyor olmasından kaynaklanmaktadır.

Şekil 2, Kaynak Kod Derleme İşlemi

Derleme Adımları

Derleme işlemi aşağıdaki adımlardan oluşmaktadır.

| # | Adım | Açıklama |
| --- | --- | --- |
| 1 | Sözcüksel analiz (ing. Lexical Analysis) | Kaynak kod içerisindeki en küçük parçaların (sayısal değerler, metin değerler, vb.) analizinin yapılması. |
| 2 | Sözcük dizim analizi (ing. Syntax Analysis) | Belirlenen sözcüklerin söz dizim kuralları kapsamında kontrol edilmesi. |
| 3 | Anlam analizi (ing. Semantics Analysis) | Sözcük dizim hataları dışındaki mantıksal hataların kontrol edilmesi. |
| 4 | Ara kod oluşturma (ing. Intermediate Code Generation) | Geçici kod üretme. |
| 5 | Optimizasyon (ing. Optimization) | Üretilen kodun optimize edilmesi. |
| 6 | Çalıştırılabilir kod oluşturma (ing. Executable Code Generation) | Optimize edilmiş geçici koddan çalıştırılabilir kodun üretilmesi. |

Tablo 3, Derleme Adımları

Yukarıda listelenen adımlar tüm derleyiciler için geçerlidir ve kod üretimi bütün adımların başarıyla tamamlanması sonucu yapılmaktadır.

Derleme işleminde yapılan ilk üç adım tüm derleyiciler için aynı niteliktedir. Bu adımların temel işlevi kaynak kod içerisinde gerek diziliş gerek mantıksal hataları yakalamak ve sonraki adımlarda işlenen kaynak kodun hatasız olduğunu garanti altına almaktır.

Ara kod olarak ifade edilen ve kaynak kod ile hedeflenen çalıştırılabilir kod arasında bulunan yapı Ara Dil Kodu – IL Kod (ing. Intermediate Language – IL Code) olarak tanımlanmaktadır. Kod üretiminde kullanılan dil, farklı derleyiciler için mantıksal düzeyde aynı olmasına rağmen, şekilsel olarak farklı üretilmektedir.

Bir derleyicinin zekâ ve değerini belirleyen en önemli unsur üretilen ara kodun optimizasyon düzeyidir. Optimizasyon işlemi, nihai çalıştırılabilen kodun kalitesini de birebir belirleyen önemli adımlardandır. Bu adımda rol oynayan en büyük etken, çalıştırılabilen kodun hedeflediği platform ve işletim sistemi özellik ve nitelikleridir. Dolayısıyla, optimizasyon işlemine başlamadan önce platform bilgisinin belirlenmiş olması en kritik değerlerden birisidir.

Azaltılmış komut kümeli mimariye sahip bilgisayarlar (ing. Reduced Instruction Set Computers – RISC), örneğin Sun Solaris, IBM Unix, vb. ile Intel tabanlı bilgisayarlar arasındaki en büyük fark, RISC tabanlı sistemlerde işlemci komut kümesinin Intel tabanlı bilgisayarlara oranla çok daha az olmasıdır. Donanımsal mimari farkının bulunduğu bu iki yapıda, derleme işlemi sonrasında oluşturulan çalıştırılabilir kodun farklı kriterler temel alınarak optimize edilmesi özellikle önem taşımaktadır.

Tamamlanan optimizasyon adımı ardından derleyicinin son görevi optimize edilmiş IL kodun çalıştırılabilir koda çevrimidir. Böylece, kullanıcı okunabilir kaynak kodun makine okunabilir koda derlenmesi tamamlanmış olmakta ve üretilen kod hedeflenen platform üzerinde çalışabilmektedir.

Yukarıdaki bilgiler ışığında, derleme işleminin tüm adımları ile tüm derleyiciler tarafından yapıldığı, işlem sırasındaki ilk üç adımın tüm derleyicilerde aynı mantık çerçevesinde çalıştığı, derleyiciler arasındaki en büyük farkın IL kod yapısı ve optimizasyon olduğu değerlendirildiğinde;  
  
Farklı programlama dillerinden aynı IL kod yapısına ulaşan bir derleyici neler sağlardı?

.NET Mimarisi ve Temelleri

Uygulama yazılımlarının işletim sisteminin sağladığı platform altında çalışabildiğinden önceki konu başlığında bahsedilmişti. Uygulama yazılımları, işletim sistemi tarafından sağlanan ve Uygulama Programlama Arabirimi (ing. Application Programming Interface - API) olarak adlandırılan hazır fonksiyonları sıklıkla kullanmaktadırlar. Bu fonksiyonlar, temel girdi – çıktı (ing. Basic input output – Basic IO) işlemlerinden hafıza yönetimine kadar farklı amaçlar için işletim sistemi üreticisi tarafından sağlanan alt seviye fonksiyonlardır.

Katman Gereksinimi

Alt seviye fonksiyon kullanımının, direkt API çağrımı, hafıza ayırma (ing. Memory Allocation) ve hafıza yok etme (ing. Memory Deallocation) işlemlerinin uygulama yazılımı tarafından yapılması gibi yöntemlere dayanıyor olması, olası hataların da ortaya çıkma ihtimalini yükseltici bir rol oynamaktadır.

**char** \*buffer = (**char**)alloc(15 \* **sizeof**(char));

printf("Gerekli hafıza ayırma yapıldı");

// Uygulamanın hata verip kapanması ...

Kod 1, Örnek Ansi C Kod Parçası

Yukarıdaki örnekte 15 karakter bilgisi alabilecek hafıza uygulama yazılımı tarafından ayrılmaktadır. Daha sonraki adımda, uygulamanın beklenilmeyen bir hata ile sonlanması sonucu ayrılan hafıza serbest bırakılmadan işlem tamamlanmaktadır.

Günümüzde kullandığımız işletim sistemi tasarımları incelendiğinde, yetkilendirmenin (ing. Authorization) kullanıcı ve rol bazlı olduğu dikkat çekmektedir. Bir başka anlatımla, yönetici kullanıcı (ing. Administrator User) tarafından çalıştırılan yazılımlar yöneticinin sahip olduğu tüm hakları miras alarak çalışmaktadırlar. Kolay kullanım perspektifinden değerlendirildiğinde avantajları olan bu yöntemin, internet ve internet üzerinden çalıştırılan yazılım kullanımının daha da artacağı varsayıldığında önemli bir güvenlik açığı oluşturabileceği değerlendirilmektedir. Kişisel bilgisayarlarına yönetici olarak giriş yapan kullanıcılar, denemek veya kullanmak istedikleri uygulama yazılımının sadece belirli dosya ve klasörlere erişimlerinin olması gibi bir dizi güvenlik kriterlerine ihtiyaç duyacaklardır.

Yukarıdaki örnekte olduğu gibi, uygulama yazılımından kaynaklı hataların artması sonucu işletim sistemi kararlılığı (ing. Stabilization) azalabilmektedir. Yönetim yöntemleri, güvenlik gereksinimleri, mimarilerin standartlaştırılması gibi bir dizi ihtiyaç ve talep de göz önüne alındığında, 1990'ların ortalarından itibaren uygulama yazılımlarının direkt işletim sistemi platformunda çalışması yerine yeni bir katman (ing. Layer) altında çalışması benimsenmiştir.

.NET Framework

Türkçe karşılığıyla .NET Çatısı olarak ifade edilebilecek .NET Framework, uygulama yazılımlarının çalışabilmesi için işletim sistemi üzerinde yeni bir katman sağlamaktan, yazılım geliştirme, dağıtma, veri işlemleri, web uygulamaları gibi bir dizi ihtiyaca standart çözümler sunan mimari (ing. Architecture) olarak tanımlanabilir.

Yazılım söz konusu olduğunda, genel prensip olarak, yazılıma yeni bir katman eklemenin sağlayacağı esneklik avantajının yanı sıra, performans düşmesi gibi önemli bir dezavantajın da ortaya çıkacağı doğaldır. Uygulama yazılımlarının .NET Framework katmanı üzerinde çalışıyor olması da bu dezavantajın beraberinde geldiğini göstermektedir. Buna rağmen farklı yöntemler ve teknolojiler yardımıyla performans düşüklüğü en aza indirilmiştir.

.NET Framework Mimarisi

.NET Framework mimarisi temel olarak;

* Farklı programlama dillerine destek.
* Ortak IL kod belirtimi (ing. Common IL Code Specification - CIL, Microsoft Intermediate Language - MSIL).
* Farklı uygulama yazılım türleri için destek ve temel sınıf kütüphanesi (ing. Class Library).
* Veri ve XML desteği.
* Çalışma zamanı (ing. Runtime) gereksinimleri için ortak platform.

Parçalarından oluşmaktadır. Aşağıda hiyerarşik yapı gösterilmiştir.

Şekil 3, .NET Framework Mimarisi

Dil Bağımsızlığı ve MSIL

.NET tarafından sağlanan en önemli avantajlardan birisi farklı programlama dillerine verilen destektir. Bir başka anlatımla, .NET platformunda çalışan uygulama yazılımlarının hangi programlama dili ile geliştirildikleri çalışma zamanında önem arz etmemektedir.

Dil bağımsızlığını sağlayan etken kitabın daha önceki bölümlerinde aktarılan derleyici yapısıdır. Microsoft tarafından IL kod belirtimi (ing. IL Code Specification) yapılmış, programlama dilinden bağımsız olarak, .NET platformunu destekleyen derleyicilerin aynı standartlarda IL kod üretimi zorunlu hale getirilmiştir.

IL kod belirtiminin yanı sıra ortak tip sistemi (ing. Common Type System - CTS) de geliştirilmiştir. Ortak tip sistemi ile nümerik, alfa-nümerik gibi tiplerin yapıları da aynı standarda kavuşturulmuştur. Böylece, programlama dili ve programlama dilindeki tip tanımları diziliş olarak farklılık gösterse de, mimari çerçevesinde tipin ne ifade ettiği sabitlenmiştir.

MSIL kodu, Yönetilen Kod (ing. Managed Code) olarak da anılmaktadır. Tersi şekilde, IL yapısında olmayan çalıştırılabilir kod da yönetilmeyen kod (ing. Unmanaged Code) olarak ifade edilmektedir.  
  
Yukarıdaki kavram genişletilerek, Yönetilen Ortam (ing. Managed Environment) ve Yönetilmeyen Ortam (ing. Unmanaged Environment) sırasıyla .NET platformu ve işletim sistemi platformu (.NET olmayan platform) için kullanılmaktadır.

Dil bağımsızlığının sağlanması ve derleyicilerin üretmiş oldukları IL kodun standartlaştırılması, nesne tabanlı yapı ve aynı zamanda aşağıda detayları aktarılan Framework sınıf kütüphanesi ile, programlama dili seçiminin ne olduğu büyük ölçüde önemini yitirmiştir.

Framework Sınıf Kütüphanesi

Framework sınıf kütüphanesi, tekrar kullanılabilir ve genişletilebilir sınıflardan oluşan kütüphanedir.

Bu kütüphane tarafından sağlanan birçok sınıf ve işlevleri uygulama yazılımları geliştirirken kullanmak mümkündür. Belirli bir hiyerarşik yapıda tasarlanan bu kütüphane ve içerdiği sınıflar hakkında detaylı bilgi kitabın ilerleyen bölümlerinde verilecektir.

Framework sınıf kütüphanesi C# programlama dili ile geliştirilmiştir.

Ortak Çalıştırma Platformu - CLR

CLR, yönetilen kodun işletim sistemi tarafından çalıştırılabilir yönetilmeyen koda dönüştürülmesi sırasında;

* Sağladığı ortak tip sistemi ile farklı programlama dillerinde geliştirilmiş yazılım parçalarının tek platformda problemsiz çalışmasından,
* Kod güvenliği.
* Hafıza yönetimi.
* Anında derleme (ing. Just in-time Compilation).
* ...

Gibi bir dizi işlemin yürütülmesinden sorumlu katman olarak tanımlanabilir.

JIT derleyici, yönetilen kodun yönetilmeyen koda derlenmesi işlemini yapmaktadır. Kitabın daha önceki konularında bahsedilen ve derleme adımlarından son ikisi olan optimizasyon ve çalıştırılabilir kod üretimi JIT tarafından yapılmaktadır.

Yönetilen kodun yönetilmeyen koda dönüşümü çalışma zamanında yapıldığına göre, geliştirilen bir uygulama yazılımının tekrar derlenmeksizin farklı donanımsal mimarilerde çalışması mümkün   
müdür ?  
  
Yani, .NET dil bağımsızlığı yanında platform bağımsızlığı da sağlamakta   
mıdır ? Lütfen Mono Projesini ([www.mono-project.com](http://www.mono-project.com)) ve DotGNU Projesini (www.dotgnu.org) inceleyiniz.

JIT tarafından yapılan derlemenin, bir derleyicinin en önemli işlevlerinden olan optimizasyon adımını içeriyor olması, aynı IL kodun farklı işletim sistemi ve donanımsal mimarilere göre çalışma zamanında optimize edilebilmesini olanaklı kılmaktadır. Böylece, son ürünün her donanım mimarisinde olabildiğince optimize edilmiş olarak çalışması sağlanmaktadır. Bu durumun, sanal bir donanım için tasarlanmış komut dizelerinin farklı donanımsal mimari özelliklerine göre çalışabilir koda dönüştürülmesinden farklı olduğu dikkate alınmalıdır.

Toplu Bakış

Aşağıda standart derleme yöntemi ve .NET Framework mimarisi ile kod üretimi şekilsel olarak gösterilmiştir.

Şekil 4, Standart Derleme Mekanizması

Şekil 5, .NET Framework Derleme Mekanizması

Sürüm Yönetimi

Microsoft.NET, C# ve VB.NET dil yapısı, Framework sınıf kütüphanesi ve CLR' de yapılan düzenleme ve eklemeleri içeren farklı sürümlere sahiptir.

Üst sürümün alt sürümü olabildiğince desteklemesi prensibi ile yapılan sürüm yönetimi, yukarıda aktarılan mimaride herhangi bir değişiklik olmaksızın, mevcut yapıların düzenlenmesi ve geliştirilmesi şeklinde olmaktadır.

Uygulama Geliştirme Araçları

.NET Framework ortak çalıştırma platformunu ve IL kod yapısını destekleyen entegre geliştirme ortamları (ing Integrated Development Environment) sayısı ve özellikleri gün geçtikçe artmaktadır.

Visual Studio.NET, Microsoft tarafından geliştirilmiş ve esnek yapısı ile .NET platformu üzerinde her türlü uygulama yazılımının rahatlıkla geliştirilebileceği araçlardan birisidir. C#, J# ve VB.NET gibi farklı programlama dillerine ve uygulama türlerine kodlama ve tasarım, test ve dağıtım (ing. Deployment) çerçevesinde destek veren VS.NET, programcıların işlerini daha az zamanda daha kaliteli yapmalarını sağlayan birçok ek özellik içermektedir.

İlk bağımsız ürün olan Borland Developer Studio ile C# ve Delphi.NET programlama dillerini kullanarak .NET platformu için uygulama yazılımları geliştirmek mümkündür. BDS.NET aynı zamanda yönetilmeyen kod üretimini de desteklemektedir. 1996 yılından bu yana Borland tarafından geliştirilen görsel bileşen kütüphanesi (ing. Visual Component Library), üçüncü parti ürün ve bileşenlerin çeşitliliği, var olan Delphi uygulamalarının platforma daha kolay taşınması gibi özellikleri ile BDS.NET ön plana çıkmaktadır.

Derleyici kavramı ile IDE kavramı çoğu zaman birbirine karıştırılmaktadır. Derleyicinin temel görevi desteklediği programlama dilinde yazılmış olan kaynak kodun derlenmesi ve çalışabilir kodun oluşturulmasıdır. IDE, görsel olarak kullanıcı arabirim tasarımı yapmaktan derleme işlemini derleyiciye yaptırmak, otomatik kaynak kod üretimi gibi daha üst seviye işlemleri yapmak, uygulama geliştirme sürecini kolaylaştırmak ve yönetmekten sorumludur. Teorik olarak, metin editörleri ve diğer araçlar kullanılarak kaynak kod programcı tarafından oluşturularak IDE kullanmadan da uygulama yazılımları geliştirmek mümkündür.  
  
Genel prensip olarak IDE, derleme işlemini ayrı bir program olarak sunulan (csc.exe, vbc.exe, dccil.exe, ...) derleyiciyi çalıştırarak ve derleyiciye gerekli parametreleri otomatik olarak geçerek yapmaktadır.

Ücretsiz olarak sunulan uygulama geliştirme araçlarının da kullanılabileceği değerlendirildiğinde, IDE seçiminde; ihtiyaç, deneyim, sunulan olanaklar gibi bir dizi kriter değerlendirilerek gerekli seçimin yapılması önerilmektedir.

Hazırlık İşlemleri ve İlk Uygulama

Bu konu başlığında, kitabın önceki bölümlerinde aktarılan kavramsal bilgilerin pratikte uygulanması ve örnek ilk uygulama hakkında bilgiler sunulacaktır.

Hazırlık İşlemleri ve Kurulumlar

.NET platformu altında çalışan uygulama yazılımları geliştirmek ve kaynak kodun derlenebilmesi için .NET Framework Yazılım Geliştirme Kiti'nin (ing. Software Development Kit) sisteme yüklenmiş olması gerekmektedir.

\SDK dizini içerisinde gerekli yazılımlar bulunabilir.

Delphi.NET kodunu derleyebilmek için lütfen Borland Corporation resmi sitesinden ([www.borland.com](http://www.borland.com)) gerekli derleyici ve IDE araçlarını indiriniz ve kurulum yönergelerine göre gerekli kurulumları yapınız.

Kaynak Kod Geliştirme

Aşağıda sırasıyla C#, Delphi.NET ve VB.NET programlama dilleri ile geliştirilmiş üç adet konsol uygulama kaynak kodu bulunmaktadır.

**using** System;

**namespace** DotNetKitabı.ÖrnekUygulamalar {

**class** Program {

**static** **void** Main(**string**[] args) {

Console.WriteLine("Herhangi bir değer giriniz:");

**string** okunanMetin = Console.ReadLine();

Console.WriteLine(

**string**.Format("Tarih-Saat:{0},Girilen Değer:{1}",

DateTime.Now.ToString(),

okunanMetin));

}

}

}

Kod 2, Örnek C# Konsol Uygulaması

**program** DotNetKitabi.OrnekUygulamalar;

**uses**

SysUtils;

**var**

OkunanMetin: **string**;

**begin**

Console.WriteLine('Herhangi bir değer giriniz:');

ReadLn(okunanMetin);

Console.WriteLine(Format('Tarih-Saat:%s, Girilen Değer:%s',

[DateTimeToStr(Now), OkunanMetin]));

**end**.

Kod 3, Örnek Delphi.NET konsol uygulaması

**Namespace** DotNetKitabı.ÖrnekUygulamalar

**Module** Program

**Sub** Main()

**Dim** okunanMetin **As** **String**

Console.WriteLine("Herhangi bir değer giriniz:")

okunanMetin = Console.ReadLine()

Console.WriteLine(**String**.Format("Tarih-Saat:{0}, \_  
 Girilen Değer:{1}", \_

DateTime.Now.ToString(), \_

okunanMetin))

**End Sub**

**End Module**

**End Namespace**

Kod 4, Örnek VB.NET Konsol Uygulaması

Listelenen kaynak kodlar \Bölüm 2\İlk Uygulama klasöründe bulunmaktadır.

Örneklerde farklı programlama dilleri kullanılarak geliştirilen kaynak kodlar incelendiğinde, Console isimli sınıfın her üç programlama dilinde de kullanıldığı görülecektir. Console sınıfı, .NET Framework Kütüphanesi tarafından sağlanan ve konsol uygulamalarında temel girdi – çıktı işlemleri için kullanılan sınıftır.

Derleme

Kurulumlar sırasında sistem veya kullanıcı dizin değişkenine (ing. Environment Path) gerekli dizin bilgileri eklenmektedir. Buna rağmen C# ve VB.NET kaynak kodlarını derlemek için kurulumda oluşturulan program grubundan SDK Command Prompt kısa yolunun kullanımı önerilmektedir.

Delphi.NET için lütfen kurulum dizini içerisindeki Bin dizinini kullanınız.

Derleme için kullanılması gereken derleyici program isimleri, parametreler ve kaynak dosya isimleri aşağıda gösterilmiştir.

csc.exe /out:konsolcs.exe /t:exe Konsol.cs

dccil.exe -CC -Xkonsoldelphi.exe Konsol.pas

vbc.exe /out:konsolvb.exe /t:exe Konsol.vb

Yukarıdaki komutların başarıyla çalışması sonucu kaynak kodlar derlenecek ve konsolcs.exe, konsoldelphi.exe ve konsolvb.exe isminde üç adet çalışabilir dosya oluşacaktır.

Çalıştırma

Konsol uygulama dosyaları üzerlerine çift tıklayarak veya komut satırı (cmd.exe) yardımıyla çalıştırınız.

Yukarıdaki adımlarda yapılan işlemler derleyici kullanmak yerine VS.NET ve BDS kullanılarak da yapılabilir. Bu durumda, uygulama geliştirme araçları tarafından yapılan yukarıda gösterilen işlemin aynısı olacaktır.

Bir sonraki bölüme geçmeden önce verilen bilgilerin kısa bir tekrarı ve gözden geçirilmesi önerilmektedir.

Mimari Analiz

Şu ana kadar aktarılan bilgiler ve bu bilgiler ışığında bir önceki bölümde geliştirilen konsol uygulamalarının bu bölümde mimari analizi yapılacaktır.

Kavramsal düzeyde aktarılan bilgilerin somut olarak gösterildiği ve bazı araçlar hakkında da bilgi verilen bu bölüm konunun daha iyi anlaşılması için oldukça önemlidir.

Derleme ve IL Kod Oluşturma

Yukarıdaki bölümde derleyiciler yardımıyla kaynak kodun IL koda dönüşümü sağlanmıştır. Üç farklı programlama dili ve üç farklı derleyici ile yapılan işlem sonucu C#, Delphi.NET ve VB.NET ile sırasıyla konsolcs.exe, konsoldelphi.exe ve konsolvb.exe adlı üç adet çalışabilir dosya oluşmuştur.

Oluşan üç adet çalışabilir dosya PE (ing. Portable Executable) veya Assebmly olarak adlandırılmaktadır.

İçerisinde IL kod bulunan ve CLR tarafından çalıştırılmaya hazır yapılara (Framework çalıştırılabilir dosyalarına) verilen isimler MSIL Kodu, Assembly ve PE şeklinde özetlenebilir. Küçük çapta bir kavram kargaşasının olduğu bu yapıda;   
  
PE, üç bölümden oluşan (PE Header, MSIL kodu ve Metadata) dosya yapısına, Assembly, bu dosyaya kavramsal olarak verilen isim olarak özetlenebilir. Farklı bir anlatımla, Assembly fiziksel yapısına verilen ad PE'dir. Temelde aynı kavramı ifade etmelerine rağmen bu yapı bundan sonra Assembly olarak anılacaktır.

MSIL Disassembler Aracı

.NET Assembly IL kodu ve içeriğini analiz etmek amacıyla Framework SDK ile birlikte dağıtılan MSIL Disassembler aracı kullanılmaktadır.

Aşağıda sırasıyla C#, Delphi.NET ve VB.NET derleyicileri ile oluşturulmuş Assembly IL kodları listelenmiştir.

.method private hidebysig static void Main(string[] args) cil managed {

.entrypoint

// Code size 46 (0x2e)

.maxstack 3

.locals init (string V\_0,

valuetype [mscorlib]System.DateTime V\_1)

IL\_0000: ldstr bytearray (48 00 65 00 72 00 68 00 61 00 6E 00 67 00 69 00 // H.e.r.h.a.n.g.i.

20 00 62 00 69 00 72 00 20 00 64 00 65 00 1F 01 // .b.i.r. .d.e...

65 00 72 00 20 00 67 00 69 00 72 00 69 00 6E 00 // e.r. .g.i.r.i.n.

69 00 7A 00 3A 00 ) // i.z.:.

IL\_0005: call void [mscorlib]System.Console::WriteLine(string)

IL\_000a: call string [mscorlib]System.Console::ReadLine()

IL\_000f: stloc.0

IL\_0010: ldstr bytearray (54 00 61 00 72 00 69 00 68 00 2D 00 53 00 61 00 // T.a.r.i.h.-.S.a.

61 00 74 00 3A 00 7B 00 30 00 7D 00 2C 00 20 00 // a.t.:.{.0.}.,. .

47 00 69 00 72 00 69 00 6C 00 65 00 6E 00 20 00 // G.i.r.i.l.e.n. .

44 00 65 00 1F 01 65 00 72 00 3A 00 7B 00 31 00 // D.e...e.r.:.{.1.

7D 00 ) // }.

IL\_0015: call valuetype [mscorlib]System.DateTime [mscorlib]System.DateTime::get\_Now()

IL\_001a: stloc.1

IL\_001b: ldloca.s V\_1

IL\_001d: call instance string [mscorlib]System.DateTime::ToString()

IL\_0022: ldloc.0

IL\_0023: call string [mscorlib]System.String::Format(string,

object,

object)

IL\_0028: call void [mscorlib]System.Console::WriteLine(string)

IL\_002d: ret

} // end of method Program::Main

Kod 5, Konsolcs.exe IL kodu

.method public static void DotNetKitabi.OrnekUygulamalar() cil managed {

.entrypoint

// Code size 83 (0x53)

.maxstack 4

.locals init (string V\_0,

object[] V\_1)

IL\_0000: ldc.i4.1

IL\_0001: stsfld bool Borland.Delphi.Units.System::IsConsole

IL\_0006: ldstr bytearray (48 00 65 00 72 00 68 00 61 00 6E 00 67 00 69 00 // H.e.r.h.a.n.g.i.

20 00 62 00 69 00 72 00 20 00 64 00 65 00 1F 01 // .b.i.r. .d.e...

65 00 72 00 20 00 67 00 69 00 72 00 69 00 6E 00 // e.r. .g.i.r.i.n.

69 00 7A 00 3A 00 ) // i.z.:.

IL\_000b: call void [mscorlib]System.Console::WriteLine(string)

IL\_0010: ldsfld class Borland.Delphi.Text Borland.Delphi.TextInput::Input

IL\_0015: call instance string Borland.Delphi.Text::ReadWideString()

IL\_001a: stloc.0

IL\_001b: ldsfld class Borland.Delphi.Text Borland.Delphi.TextInput::Input

IL\_0020: call instance void Borland.Delphi.Text::ReadLn()

IL\_0025: call void Borland.Delphi.Units.System::@\_IOTest()

IL\_002a: ldstr bytearray (54 00 61 00 72 00 69 00 68 00 2D 00 53 00 61 00 // T.a.r.i.h.-.S.a.

61 00 74 00 3A 00 25 00 73 00 2C 00 47 00 69 00 // a.t.:.%.s.,.G.i.

72 00 69 00 6C 00 65 00 6E 00 20 00 44 00 65 00 // r.i.l.e.n. .D.e.

1F 01 65 00 72 00 3A 00 25 00 73 00 ) // ..e.r.:.%.s.

IL\_002f: ldc.i4.2

IL\_0030: newarr [mscorlib]System.Object

IL\_0035: stloc.1

IL\_0036: ldloc.1

IL\_0037: ldc.i4.0

IL\_0038: call valuetype Borland.Delphi.TDateTime Borland.Delphi.TDateTime::Now()

IL\_003d: call string Borland.Vcl.Units.SysUtils::DateTimeToStr(valuetype Borland.Delphi.TDateTime)

IL\_0042: stelem.ref

IL\_0043: ldloc.1

IL\_0044: ldc.i4.1

IL\_0045: ldloc.0

IL\_0046: stelem.ref

IL\_0047: ldloc.1

IL\_0048: call string Borland.Vcl.Units.SysUtils::Format(string,

object[])

IL\_004d: call void [mscorlib]System.Console::WriteLine(string)

IL\_0052: ret

} // end of method OrnekUygulamalar::DotNetKitabi.OrnekUygulamalar

Kod 6, Konsoldelphi.exe IL kodu

.method public static void Main() cil managed {

.entrypoint

.custom instance void [mscorlib]System.STAThreadAttribute::.ctor() = ( 01 00 00 00 )

// Code size 46 (0x2e)

.maxstack 3

.locals init (string V\_0,

valuetype [mscorlib]System.DateTime V\_1)

IL\_0000: ldstr bytearray (48 00 65 00 72 00 68 00 61 00 6E 00 67 00 69 00 // H.e.r.h.a.n.g.i.

20 00 62 00 69 00 72 00 20 00 64 00 65 00 1F 01 // .b.i.r. .d.e...

65 00 72 00 20 00 67 00 69 00 72 00 69 00 6E 00 // e.r. .g.i.r.i.n.

69 00 7A 00 3A 00 ) // i.z.:.

IL\_0005: call void [mscorlib]System.Console::WriteLine(string)

IL\_000a: call string [mscorlib]System.Console::ReadLine()

IL\_000f: stloc.0

IL\_0010: ldstr bytearray (54 00 61 00 72 00 69 00 68 00 2D 00 53 00 61 00 // T.a.r.i.h.-.S.a.

61 00 74 00 3A 00 7B 00 30 00 7D 00 2C 00 20 00 // a.t.:.{.0.}.,. .

47 00 69 00 72 00 69 00 6C 00 65 00 6E 00 20 00 // G.i.r.i.l.e.n. .

44 00 65 00 1F 01 65 00 72 00 3A 00 7B 00 31 00 // D.e...e.r.:.{.1.

7D 00 ) // }.

IL\_0015: call valuetype [mscorlib]System.DateTime [mscorlib]System.DateTime::get\_Now()

IL\_001a: stloc.1

IL\_001b: ldloca.s V\_1

IL\_001d: call instance string [mscorlib]System.DateTime::ToString()

IL\_0022: ldloc.0

IL\_0023: call string [mscorlib]System.String::Format(string,

object,

object)

IL\_0028: call void [mscorlib]System.Console::WriteLine(string)

IL\_002d: ret

} // end of method Program::Main

Kod 7, Konsolvb.exe IL Kodu

Üç Assembly tipi bulunmaktadır. Bunlar konsol uygulaması (.exe), Windows uygulaması (.exe) ve kütüphane (.dll) olarak özetlenebilir. Derleyiciye iletilen parametreler yardımıyla hangi tip Assembly üretileceği belirlenmektedir (ör. /T:winexe).  
  
Assembly dosya uzantılarının standart Windows uygulamaları ile aynı olmaları, bu dosyaların, içerisinde makine kodu içeren Windows çalıştırılabilirleri ile karıştırılmasına sebep olmamalıdır.

Çalıştırma ve CLR

İşletim sistemine çalıştır komutu verildikten sonra rutin uygulama çalıştırma işlemi başlamaktadır.

Arka planda, geneli itibariyle, sırasıyla aşağıdaki işlemler yapılmaktadır.

1. Fiziksel dosya yapısı standart Windows çalışabilir (PE) yapısında olduğu için ilk adım sistem üzerinde .NET CLR'nin hafızaya yükleme işleminin yapılmasıdır. Bu adımdan sonraki tüm işlemler CLR tarafından yapılıyor olacaktır.
2. PE yapısında olan dosyalar Metadata adı verilen bir bölüm içermektedir. Bu bölüm CLR tarafından analiz edilerek gerekli kontroller ve sınıf, nesne yüklemeleri yapılır.
3. Assembly içerisinde bulunan ve daha önceki bölümde gösterilen IL kodu JIT derleyici tarafından derlenir ve makine kodu oluşturulur.
4. Makine kodu işletim sistemi platformunda çalıştırılır.

Yansıma

İngilizce karşılığı Reflection olan yansıma yardımıyla, derleyici tarafından oluşturulan IL kodun tekrar kaynak koda dönüştürülmesi mümkündür. Güvenlik ve gizlilik kriterleri değerlendirildiğinde irdelenmesi gereken önemli bir konu olarak değerlendirilen yansıma için çeşitli hazır yazılımlar kullanmak mümkündür.

IL kod üzerinde çalışıldığı için, yansıma kullanımında Assmbly'nin hangi programla dili ile geliştirildiği önem arz etmemektedir. Örneğin C# dilinde geliştirilmiş kaynak koddan üretilen bir Assembly IL kodunun Delphi.NET diline çevrimi mümkündür.

Aşağıda, bu işlemi örneklemek için Lutz Roader tarafından geliştirilen yazılımın ekran çıktısı gösterilmiştir.

Ekran Çıktısı 1, Lutz Roader .NET Reflector Aracı

Kaynak kod hakkında bilgi almak için ters dönüşüm işlemi önemli avantajlar sağlamaktadır. Özellikle Framework sınıf kütüphanesi dokümantasyon bilgilerinin yetersiz olduğu durumlarda işlemlerin nasıl yapıldığı konusunda bu avantaj sonuna kadar kullanılmalıdır.

\Yardımcı Araçlar\Lutz\reflector.zip dosyasında ters dönüşüm işlemini farklı dillere göre yapan uygulama bulunabilir.

Kavramlar

Bu bölümde, kitabın ilerleyen bölümlerinde kullanılacak olan kavramların tanımları ve açıklamaları bulunmaktadır.

Kavramlar ve Tanımları

ADO.NET

Framework sınıf kütüphanesi içerisinde, veri ve veritabanı işlemlerinin bulunduğu katmana verilen isimdir.

ASP.NET

Framework içerisinde web uygulama ve hizmetleri geliştirme ve çalıştırma katmanına verilen isimdir.

Global Assembly Cache (GAC)

Sistemde farklı uygulamaların paylaşabileceği Assembly'lerin merkezi olarak tutulduğu yapıdır. Kontrol paneli (ing. Control Panel) içerisinde bulunan .NET Konfigürasyon aracı ile GAC yönetimi yapılabileceği gibi gacutil.exe ile de aynı yönetim yapılabilir.

Garbage Collector (GC)

Hafıza yönetimi CLR tarafından otomatik olarak yapılmaktadır. Hafızada bulunan ve artık kullanılmayan bölümlerin belirli zaman aralıkları ile temizlenmesinden GC sorumludur.

HyperText Markup Language (HTML)

Web görselliğini tanımlayan dildir.

Namespace

Ortak tiplerin mantıksal organizasyonu için kullanılan yapıdır.

Bir Assembly, içerisinde birden çok Namespace içerebilir.

Simple Object Access Protocol (SOAP)

XML yapısında nesnelerin taşınabilmesini sağlayan protokole verilen isimdir.

Strong Name

Assembly'ye verilen ismin sürüm, kültür ve açık anahtar ile tekilleştirilmesini sağlayan isimlendirme standardıdır.

Universal Description, Discover, and Integration (UDDI)

Web hizmetlerinin belirli kurallar çerçevesinde bulunmasını sağlayabilen yapıdır.

Extensible Markup Language (XML)

Veri ve veri yapısını standart kurallar ile tanımlayabilen dildir.

Relational Database Management System – RDBMS

İlişkisel veritabanı yönetim sistemidir. SQL komutları ile veriye erişim ve ver manipülasyonu sağlayan, çok kullanıcı ortamlarda veri güvenliği ve performans kazanımlarını sunan sunucu yazılımına verilen isimdir.

Windows Forms.NET

.NET sınıf kütüphanesi içerisinde Windows işletim sistemi altında çalışabilecek pencere, kontrol gibi yapıları içeren kütüphaneye verilen isimdir.

Ne kadar bilirsen bil söylediklerin karşındakinin anladığı kadardır.

Mevlâna Celâleddin-i Rûmî

Bölüm

3

Yazılım Geliştirme Ortamları

E

ntegre Geliştirme Ortamı – IDE, programcıların yazılım geliştirme süreçlerinde, kaynak kod geliştirme adımından test süreçlerine kadar, otomatik kod üretimi, kod analizi, test senaryolarının oluşturulması gibi zaman ve emekten önemli kazanımlar elde etmeyi hedefleyen uygulama yazılımı olarak tanımlanabilir.

Derleyici ve IDE, birbirlerine karıştırılabilen kavramlardır. Derleyicinin temel amacı kaynak kodu derlemek iken, IDE kullanımı ile hedeflenen, kaynak kod geliştirme ve diğer yazılım süreçlerinde kaliteyi artırmak, zamandan kazanmaktır.

Günümüzde yazılım geliştirirken IDE kullanımı neredeyse zorunlu hale gelmiştir. Görsel tasarımların yapılması, otomatik kod üretimi gibi işlevlerde IDE kullanılmadan programlama büyük emek ve zaman kaybı anlamına gelmektedir.

IDE tarafından bir çok işlemin otomatik olarak yapılıyor ve görsel tasarımın kolaylaşmış olması, programcıların arka planda yapılan işlemleri bilmemeleri ile birleştiğinde önemli teknik problemlerin de önü açılmaktadır. Mimari ve kavramsal düzeyde de olsa arka planda yapılan işlemlerin neler olduğunun bilinmesi büyük önem taşımaktadır.

Visual Studio.NET ve Borland Developer Studio, .NET platformunda uygulama yazılımı geliştirmeyi hedefleyen iki önemli IDE olarak ifade edilebilir. Temelde işlevleri birbirlerine oldukça benzemekte olan bu iki ürünün kullanım detaylarında bazı farklar bulunmaktadır. Kitabın bu bölümünde VS.NET ağırlıklı olarak genel kullanım ve özellikler hakkında bilgiler verilecektir.

Hazırlık İşlemleri ve Kurulumlar

VS.NET, Microsoft tarafından .NET platformunda uygulama geliştirmek için kullanıma hazırlanmış yazılım geliştirme ortamıdır. Ticari olarak farklı sürüm ve özelliklerde sunulan VS.NET hakkında Microsoft firmasının internet sitesinden ([www.microsoft.com.tr](http://www.microsoft.com.tr)) ticari sunum ve lisanslama konularında bilgi edinilebilir.

Ürünün kurulum detayları bu kitabın kapsamı dışındadır. Ürün kurulumu gerekli kurulum yönergeleri yardımıyla yapıldıktan sonra ilgili program grubundan çalıştırılabilir.

Genel Kavramlar

VS.NET, farklı proje tipleri ve proje özelliklerine entegre olabilen esnek bir yapıda geliştirilmiştir. Ürün ilk kurulduğunda ürün ile birlikte gelen proje şablonlarına ek olarak üçüncü firma ve kuruluşların ürettikleri ürünler de VS.NET ile rahatlıkla entegre olarak tek bir ürünmüşçesine çalışabilmektedir.

Aşağıda temel VS.NET kavramları ve açıklamaları listelenmiştir.

| Kavram | Açıklama |
| --- | --- |
| Çözüm (ing. Solution) | İçerisinde farklı proje ve proje tiplerini barındırabilen yapıdır. |
| Proje (ing. Project) | Farklı amaçlara yönelik işlem yapmayı hedefleyen, çoğunlukla çıktı üreten, kendi içerisinde bütün olan yapıdır. |
| Proje Tipi (ing. Project Type) | Yazılım projelerinden veritabanı projelerine kadar, projenin tipini ifade etmektedir. |
| Proje Şablonu (ing. Project Template) | Proje geliştirirken başlangıç noktası olarak kullanılabilecek hazır yapıdır. |

Tablo 4, Temel VS.NET kavramları

Aşağıda VS.NET genel görünümü gösterilmiştir.

Ekran Çıktısı 2, VS.NET genel görünümü

VS.NET, kullanım ve ulaşımı kolaylaştırmak ve farklı işlevleri yönetebilmek için kendi içerisinde pencereler içermektedir.

Toolbox, görsel tasarım sırasında kullanılabilecek kontrol ve bileşenleri içeren penceredir. Solution Explorer, proje içeriklerinin yönetimi için kullanılabilecek komutlar ve genel hiyerarşiyi gösterir pencere olarak tanımlanabilir. Seçili nesnenin otomatik olarak özelliklerini görüntülemek ve düzenlemek için Properties penceresi kullanılmalıdır.

Temel Çözüm ve Proje Bilgileri

VS.NET ile farklı programlama dilleri kullanılarak uygulama yazılımları geliştirilebilmektedir. Aynı zamanda kurulum, veritabanı veya iş zekası (ing. Business Intelligence) gibi projeler de desteklenmektedir.

Bir yazılım projesi, içerisinde farklı mantıksal katmanlar içerebileceği gibi farklı amaçlar için geliştirilmiş farklı uygulama türleri de içerebilir. Örneğin komple çözüm sunan bir yazılım projesi değerlendirildiğinde, internet kullanıcılarına hizmet veren bir web sitesi, yönetimsel işlemlerin yapılabileceği Windows uygulaması, aynı zamanda mobil cihazlar ile kullanılabilen ayrı bir kullanıcı arabirimi bu projenin çeşitli parçaları olabilir. Farklı proje tiplerini (örneğin sırasıyla Web uygulaması, Windows uygulaması ve Windows CE uygulaması) içerebilen böyle bir kapsayıcıya VS.NET terminolojisinde verilen isim çözümdür (ing. Solution).

Çözüm, içerisinde farklı projeler içerebilir ve projelerin birbirlerine olan bağımlılıkları VS.NET tarafından otomatik olarak yönetilebilir.

VS.NET içerisinde bulunan çözüm gezgini (Solution Explorer) ve seçili nesne özellikleri (Properties) penceresi yazılım geliştirirken en çok kullanılan işlevlerin yönetiminin yapılacağı komutlar içermektedir.

Dosya Tipleri ve Uzantıları

Aşağıda sık kullanılan bazı dosya tipleri, uzantıları ve kısa açıklamaları gösterilmiştir.

| Dosya | Uzantı | Açıklama |
| --- | --- | --- |
| Çözüm dosyası | .sln | VS.NET çözüm dosyasıdır. |
| Çözüm kullanıcı ayarları | .suo | Bir çözümle ilişkilendirilen kullanıcı bazlı ayarlama ve tanımlamaların tutulduğu dosyadır. |
| C# kaynak kod dosyası | .cs |  |
| VB.NET kaynak kod dosyası | .vb |  |
| J# kaynak kod dosyası | .jsl |  |
| Veri kümesi (ing. Dataset) | .xsd | Veri kümesi mantıksal yapısını tutan dosyadır. |
| Kaynak dosyası (ing. Resource file) | .resx | Uygulamaya ait çeşitli kaynakları (resim, metin değer, vb.) tutan dosyadır. |
| Hata ayıklama (ing. Debug) bilgileri dosyası | .pdb | Assembly hata ayıklama bilgilerini tutan dosyadır. |
| C# projesi | .csproj |  |
| VB.NET projesi | .vbproj |  |
| J# projesi | .jsproj |  |

Tablo 5, VS.NET dosya uzantı açıklamaları

Çözüm ve Proje Yönetimi

Çözüm ve projelerin en etkin şekilde yönetimi bu konu başlığında aktarılacaktır.

Çözüm Yönetimi

Daha önce de aktarıldığı üzere çözüm, içerisinde bir veya daha fazla proje içerebilen mantıksal kapsayıcıdır.

VS.NET ile boş bir çözüm yaratmak için yeni proje penceresinde Other Project Types | Visual Studio Solutions bölümünden Blank Solution komutu verilmelidir.

Ekran Çıktısı 3, Yeni bir çözüm yaratılması

Name, çözüme verilen adı, Location çözüm dosya klasörünü ifade etmektedir. Yukarıdaki işlemin başarıyla tamamlanması ile yaratılan çözüm VS.NET çözüm gezgini penceresinde gözükecektir.

Ekran Çıktısı 4, Çözüm gezgini penceresi

Klasör yapısı incelendiğinde, belirtilen klasör içerisine DotnetKitabı isminde yeni bir klasör açıldığı ve bu klasör içerisinde DotnetKitabı.sln ve DotnetKitabı.suo isimli iki adet VS.NET çözüm dosyasının bulunduğu görülecektir.

Çözüm Klasörleri

Çözümler, çözüm ile ilişkilendirilebilen ve mantıksal organizasyonu bir ileri seviyeye götürebilmek için kullanılabilen çözüm klasörleri içerebilirler.

Yeni bir çözüm klasörü eklemek için çözüm gezgini penceresinde ilgili çözüm üzerine sağ tıklayarak Add | New Solution Folder komutu verilmelidir.

Ekran Çıktısı 5, Çözüm klasörü eklemek

Çözüm klasörleri sanal klasörlerdir ve yeni bir çözüm klasörü eklendiğinde dosya sisteminde bunun için yeni bir klasör yaratılmaz. Çözüm mantıksal organizasyonu ile disk üzerindeki dosya ve klasör yerleşimin uyumlu olması için çözüm dosyasının bulunduğu klasör içerisine aynı isimli klasör yaratılması ve bu klasör içerisine dosyaların eklenmesi önerilmektedir.

Çözüm İçerik Yönetimi

Çözüm veya çözüm klasörü üzerine sağ tıklanarak açılan Add menüsü aracılığıyla çözüme yeni içerikler eklenebilir.

Aşağıda temel komutlar ve açıklamaları gösterilmiştir.

| Komut | Açıklama |
| --- | --- |
| New Project | Çözüme yeni bir proje eklemek için kullanılmalıdır. Bu durumda yeni proje penceresi açılacaktır. |
| Existing Project | Var olan bir projenin çözümle ilişkilendirmesini ve çözüme eklemek için kullanılmalıdır. |
| New Web Site | Çözüme yeni bir web sitesi eklemek için kullanılmalıdır. |
| Existing Web Site | Çözüme var olan web sitesini eklemek için kullanılmalıdır. |
| New Item | Çözüme proje ve web sitesi dışında çeşitli içerik (metin, resim dosyası, vb.) eklemek için kullanılmalıdır. |
| Existing Item | Çözüme var olan herhangi bir içerik eklemek için kullanılmalıdır. |

Tablo 6, Çözüm içerik tipleri

Çözüme daha önce eklenmiş bir içeriğin çözümden çıkartılması için ilgili içerik üzerine sağ tıklanarak Remove komutu verilmelidir.

Remove komutu ile içeriğin çözüm içerisindeki referansı kaldırılmakta, içerik fiziksel olarak diskten silinmemektedir.

Çözüm Özellikleri Yönetimi

Çözüm üzerinde sağ tıklanarak çözüme ait mantıksal özellikler görüntülenebilir veya düzenlenebilir.

Proje Yönetimi

Yeni bir proje oluşturmak için File | New | Project... veya çözüm gezgini penceresinde çözüm üzerine sağ tıklanarak Add | New Project komutu verilmelidir.

Daha önce üzerinde çalışılan bir çözüm açık iken File | New | Project ... komutu verildiğinde açılan yeni proje penceresinde projenin var olan çözüme eklenmesi veya yeni proje ile yeni bir çözüm yaratılıp yaratılmayacağına dair bilgi alınmaktadır.

VS.NET ile çalışırken her projenin bir çözümle ilişkilendirilmesi gerekmektedir. Ancak, proje dosyası içerisinde projenin hangi çözüme ait olduğu değil, çözüm dosyasında proje dosyasına referans bulunmaktadır. Böylece bir projenin farklı çözümlerde yer alabilmesi mümkün olmaktadır.  
  
Çözüm, içerisinde bir proje içeriyor ise çözüm gezgininde sadece proje ismi gözükmektedir. Bunu engellemek ve çözümün de her durumda gözükmesini sağlamak için Tools | Options penceresinde Projects and Solutions sekmesinde Always Show Solution seçeneği işaretli olmalıdır.

Ekran Çıktısı 6, VS.NET ile yeni Konsol Uygulaması Eklenmesi

Yukarıdaki işlemin başarıyla tamamlanması ile daha önce yaratılan çözüme konsol uygulaması tipinde ve C# dilinde yeni bir proje eklenmiş olacaktır.

Aşağıda çözüm gezgini ve fiziksel disk hiyerarşisi gösterilmektedir.

Ekran Çıktısı 7, Çözüm Gezgini Konsol Uygulaması

Proje isimlendirme standardında, ön ek olarak çözüm ismi kullanımı önerilmektedir.

Ekran Çıktısı 8, Çözüm ve proje fiziksel organizasyonu

Proje Özellikleri

Çözüm gezgini penceresinde ilgili proje seçilip sağ tıklanıp Properties komutu verilerek projeye ait özellikler görüntülenebilir veya düzenlenebilir. Farklı proje tipleri için farklı özelliklerin düzenlenebileceği bu ekran hakkında kitabın farklı bölümlerinde detaylı bilgiler verilecektir.

Temel İşlemler ve Açıklamaları

Aşağıda proje ile ilgili temel komutlar ve açıklamaları listelenmiştir.

| Komut | Açıklama |
| --- | --- |
| Refresh | Proje içeriğinin güncellenmesi için kullanılmaktadır. |
| Build / Rebuild | Projenin tamamını derlemek amacıyla kullanılmaktadır. |
| Clean | Proje geçici dosya ve çıktıların temizlenmesi için kullanılmalıdır. |
| Add … | Projeye yeni içerik eklemek için kullanılmalıdır. |
| Set As Startup Project | Projenin, çözümün başlangıç projesi olmasını sağlar. |
| Debug | Start New Instance | Hata ayıklamak için ilgili projenin çalıştırılmasını sağlar. |
| Add Reference … | Projeye yeni bir Assembly referansı eklemek için kullanılmalıdır. |

Tablo 7, Proje Temel İşlemleri ve Açıklamaları

Yukarıda listelenmiş olan komutlar farklı proje tipleri için değişiklik gösterebilir.

Derleme ve Çalıştırma

VS.NET ile tüm çözümü veya herhangi bir projeyi derlemek mümkündür.

Tüm çözümü, yani çözüm içerisindeki tüm projeleri derlemek için, çözüm gezgini penceresi içerisindeki çözüm üzerine sağ tıklayarak Build veya Rebuild komutunu vermek veya Ctrl + Shift + B tuş takımını kullanmak mümkündür.

Ekran Çıktısı 9, Çözümün Derlenmesi

Build ve Rebuild komutları birbirinden farklıdır.  
  
Build komutu verildiğinde, son başarılı derlemeden sonra üzerinde değişiklik yapılmamış olan projeler derlenmemektedir. Rebuild komutu verildiğinde ise üzerinde değişiklik yapılıp yapılmadığına bakılmaksızın tüm projeler derlenmektedir.

Benzer şekilde, belirli bir projeyi derlemek için proje üzerinde sağ tıklayarak Build veya Rebuild komutunu vermek yeterlidir.

Bir çözüm bir veya birden çok başlangıç projesi (ing. Startup Project) içerebilir. Bir proje üzerinde sağ tıklayarak Set As Startup Project komutu verilerek ilgili proje başlangıç projesi yapılabilir.

Debug | Start Debugging komutu ile veya F5 tuşu ile başlangıç projelerinin çalıştırma işlemi başlatılabilir.

Hata ayıklama işlemi yapmadan çalıştırmak için Debug | Start Without Debugging komutu verilmeli veya Ctrl + F5 tuş takımı kullanılmalıdır.

İleri Seviye İşlemler

Yazılım geliştirme sürecinde, kullanılan IDE tercihi ne olursa olsun hız ve alışkanlık kazanmak için en önemli kriter ürün üzerinde sahip olunan deneyimdir. Bu deneyimin ürünü kullandıkça zamanla kazanılacağı göz önüne alınmalıdır.

Bu bölümde VS.NET konusunda daha önceki bölümlerde verilen temel kavramlar ve kullanım detayları ışığında daha ileri seviye işlemler hakkında bilgiler sunulacaktır.

Verilen bilgiler hakkında kavramsal düzeyde bilgisi olmayan okuyucular için bir sonraki bölümü okuduktan sonra bu bilgiler daha iyi anlaşılmış olacaktır.

Hazırlık İşlemleri

Bu bölümde anlatım, örnek bir VS.NET çözümü odaklı sürdürülecektir. Bir önceki bölümde hazırlanmış olan çözüme DotnetKitabi.WindowsUygulaması, Dotnet.WebUygulaması ve DotnetKitabi.OrtakIslemler isimli üç adet sırasıyla Windows uygulaması, ASP.NET Web uygulaması ve sınıf kütüphanesi tipinde üç adet proje eklenmiştir.

Nihai hedeflenen bir sınıf kütüphanesi projesini ortak olarak kullanan üç adet farklı uygulama tipinin örneklenmesi ve bu örnek üzerinden VS.NET işlemlerinin aktarımının yapılmasıdır.

Bölüm 3\DotnetKitabı klasöründe bu bölümde geliştirilen kaynak kodlar bulunmaktadır.

Şekil 6, VS.NET Çözüm ve 4 Projenin Kavramsal Gösterimi

Windows Uygulaması Projesi Ekleme

Çözüm üzerine sağ tıklayarak Add | New Project komutu veriniz.

Ekran Çıktısı 10, Yeni Windows Uygulaması Projesi Eklenmesi

Web Uygulaması Projesi Eklenmesi

Çözüm üzerine sağ tıklayarak Add | New Project komutu veriniz.

Ekran Çıktısı 11, Yeni Web Uygulaması Projesi Eklenmesi

VS.NET 2005 SP1 ve üzeri sürümlerde desteklenmektedir. Alt sürümlerde Add | New Web Site komutu verilmelidir.

Sınıf Kütüphanesi Projesi Eklenmesi

Çözüm üzerine sağ tıklayarak Add | New Project komutu veriniz.

Ekran Çıktısı 12, Yeni Sınıf Kütüphanesi Projesi Eklenmesi

Genel Görünüm

Aşağıda dört adet farklı nitelikte proje içeren çözüm gezgini penceresi gösterilmiştir.

Ekran Çıktısı 13, Projeler ve Çözüm Gezgini Penceresi

Proje Referansları Yönetimi

Bir proje derlendiğinde çıktı olarak proje adında ve proje niteliğine göre uzantısı belirlenen bir Assembly oluşturulacaktır.

Assembly adı ve proje tipi proje özellikleri penceresi yardımıyla düzenlenebilir.

Ekran Çıktısı 14, Konsol Uygulaması Proje Özellikleri Penceresi

VS.NET farklı proje tiplerini ve proje tiplerine göre farklı çıktıları desteklediği için proje özellikleri penceresi proje tipine göre farklılıklar içermektedir.

Projenin başka bir projeyi veya Assembly dosyasını kullanabilmesini sağlamak için proje üzerinde sağ tıklanarak Add Reference komutu verilmelidir.

Ekran Çıktısı 15, Projeye Yeni Referans Eklenmesi

Referans ekleme penceresi ile GAC içerisinde bulunan Assembly, COM sınıfları, çözüm içerisindeki projeler veya herhangi bir Assembly dosyasına referans vermek mümkündür.

İşlemin başarıyla tamamlanması ardından eklenen referans proje referansları bölümünde otomatik olarak görüntülenmektedir.

Ekran Çıktısı 16, Proje Referansları

Referansın eklenmesi ile birlikte, DotnetKitabı.Ortakİşlemler projesi ile üretilen DotnetKitabı.Ortakİşlemler.dll isimli Assembly içerisindeki sınıf ve fonksiyonlar konsol uygulamasında kullanılabilir hale gelmiştir.

Referans özelliklerinden Copy Local, eklenen referans ile ifade edilen Assembly dosyasının proje çıktı klasörüne kopyalanıp kopyalanmayacağını belirtmektedir. Değerinin True olması durumunda referans proje çıktı klasörüne kopyalanır, False olması durumunda kopyalanmaz.

Mevcut referansı kaldırmak için üzerine sağ tıklayıp Remove komutu verilmelidir.

Örnek çözümde hedeflenen, sınıf kütüphanesinin diğer projeler tarafından kullanılması olarak planlandığı için, yukarıda konsol uygulaması için yapılan işlem diğer projelerde de yapılmalıdır.

Çözüm Özellikleri Yönetimi

Çözüm gezgini penceresi içerisinde, çözüm üzerinde sağ tıklayarak Properties komutu verilip çözüme ait özelliklerin görüntülenebileceği veya düzenlenebileceği pencere açılabilir.

Ekran Çıktısı 17, Çözüm Özellikleri Penceresi

Başlangıç Projeleri Tanımı

Startup Project sekmesi yardımıyla çözüm çalıştırılmak istendiğinde hangi proje veya projelerin çalıştırılacağı belirtilebilir.

Current Selection seçeneği, çözüm gezgini penceresinde seçili olan projenin çalıştırılması gerektiğini belirtmektedir. Single Startup Project seçeneği ile çözüm içerisinde bulunan herhangi bir proje, Multiple Startup Projects seçeneği ile birden fazla proje başlangıç projesi olarak tanımlanabilir.

Proje Bağımlılıkları Yönetimi

Project Dependencies sekmesi yardımıyla çözüm içerisinde bulunan projelerin birbirlerine olan bağımlılıkları yönetilebilir.

Bir projenin bir başka projeye bağımlı olması, ilk projeden diğer projeye bir referans olması anlamına gelmektedir. Üzerinde çalışılan örnekte, konsol uygulaması, Windows uygulaması ve web uygulaması, sınıf kütüphanesi projesine bağımlıdır. Sınıf kütüphanesinin herhangi bir bağımlılığı bulunmamaktadır.

Çözüm üzerinde sağ tıklanıp Project Build Order komutu verildiğinde çözümün derlenmesi işleminde derlemenin hangi sırada yürütüleceğine dair bilgi görüntülenebilir.

Hata Ayıklama Kaynak Dosya Yönetimi

Debug Source Files, hata ayıklama işlemi sırasında kaynak kod araması yapılması istenen klasörlerin belirtildiği bölümdür. VS.NET, hata ayıklama sırasında kullanılan Assembly ile kaynak kod dosyasını eşleştiremediğinde bu bölümde belirtilen klasörlerde kaynak kod dosyasını aramaktadır.

Konfigürasyon Özellikleri Yönetimi

VS.NET ile gerek çözüm gerek proje bazında farklı konfigürasyonlar tanımlamak mümkündür.

Ekran Çıktısı 18, Çözüm Konfigürasyon Sekmesi

Konfigürasyon, çözümün derlenmesi sırasında çözüm içerisinde bulunan projelerin nasıl ele alınacağını belirten kavramdır. Debug konfigürasyon tanımı, hata ayıklama amacıyla derleme işleminin yapılacağını, Release konfigürasyon tanımı ise hata ayıklama bilgileri olmadan derleme işleminin yapılacağını belirtmektedir.

Proje bazında Build seçeneğinin seçilmesi, ilgili projenin çözüm derleme işleminde derlenmesi gerektiğini, işaretin seçili olmaması ise projenin derlenmemesi gerektiğini belirtmektedir.

Build ve Debug konfigürasyon tanımları yanı sıra konfigürasyon yöneticisi (ing. Configuration manager) penceresi yardımıyla yeni konfigürasyon tanımları yapmak mümkündür.

Tasarım ve Geliştirme

Hedeflenen mimarinin daha anlamlı olması için MaaşHesapla isimli bir metot DotnetKitabı.Ortakİşlemler isimli projeye eklenecektir.

Ekran Çıktısı 19, Projeye Yeni Sınıf Eklenmesi

**namespace** DotnetKitabı.Ortakİşlemler {

**public** **class** Maaşİşlemleri {

**public** **static** **int** MaaşHesapla(**int** katSayı) {

**return** katSayı \* 1000;

}

}

}

Kod 8, MaaşHesapla Metodu

Konsol Uygulaması İşlemleri

Konsol uygulaması içerisindeki program.cs isimli dosya içeriği aşağıda gösterilmiştir.

**using** DotnetKitabı.Ortakİşlemler**;**

**namespace** DotnetKitabı.KonsolUygulaması {

**class** Program {

**static** **void** Main(**string**[] args) {

Console.WriteLine(Maaşİşlemleri.MaaşHesapla(5));

}

}

}

Kod 9, Konsol Uygulaması İçeriği

Yukarıdaki kod parçası incelendiğinde using bloğu içerisinde DotnetKitabı.Ortakİşlemler isimli Assembly tanımının bulunduğu görülecektir.

Windows Uygulaması İşlemleri

Görsel tasarımı yapılmış örnek form aşağıda gösterilmiştir.

Ekran Çıktısı 20, Windows Formu

Hesapla başlıklı butona tıklandığında çalışacak olan kod parçası aşağıda gösterilmiştir.

**private** **void** ctlHesapla\_Click(**object** sender, EventArgs e) {

ctlSonuç.Text =   
 Maaşİşlemleri.MaaşHesapla(**int**.Parse(ctlKatsayı.Text)).ToString();

}

Kod 10, Windows Uygulaması Örnek Kod Parçası

Web Uygulaması İşlemleri

Görsel tasarımı yapılmış örnek form aşağıda gösterilmiştir.

Ekran Çıktısı 21, Web Formu

Hesapla başlıklı butona tıklandığında çalışacak olan kod parçası aşağıda gösterilmiştir.

**private** **void** ctlHesapla\_Click(**object** sender, EventArgs e) {

ctlSonuç.Text =   
 Maaşİşlemleri.MaaşHesapla(**int**.Parse(ctlKatsayı.Text)).ToString();

}

Kod 11, Web Uygulaması Örnek Kod Parçası

Kod parçalarından da anlaşılacağı üzere, farklı uygulama türlerinde tasarım farkı dışında kavramsal düzeyde yapılan işlemler, hatta kullanılan kontrollerin özellik isimleri dahi aynıdır.

Çalıştırma ve Hata Ayıklama

VS.NET ile bir çözümün çalıştırılması, çözüm konfigürasyon bilgilerinde tanımlı olan projelerin çalıştırılması anlamına gelmektedir.

Çalıştırma, iki şekilde başlatılabilir. F5 tuşu ile çalıştırma hata ayıklamak üzere işlemin yapıldığını, Ctrl + F5 tuş kombinasyonu ile çalıştırma hata ayıklama işlemi olmadan projelerin çalıştırılması gerektiğini ifade etmektedir.

Üzerinde çalışılan örnekte aşağıda belirtilen çözüm konfigürasyonu ile F5 tuşuna basıldığında konsol uygulaması ve Windows uygulaması otomatik olarak çalıştırılacaktır.

Ekran Çıktısı 22, Örnek Çözüm Konfigürasyonu

Hata Ayıklama İşlemi

Uygulama yazılımları geliştirirken yazılımda hata ayıklama işlemi en önemli adımlardan birisi olarak değerlendirilebilir.

İngilizce tanımı Debugging olan hata ayıklama işlemi, derleme işlemi ile ilgilidir. Derleme sırasında derleyici tarafından hata ayıklama bilgilerini içeren ve uzantısı .pdb (Program Database) olan dosyalar hata ayıklama amacıyla oluşturulmaktadır.

Durma noktası olarak tanımlanabilecek Breakpoint kavramı, kodun işletimi sırasında yazılımın durdurulmasını ve VS.NET hata ayıklayıcısının aktif olmasını sağlamaktadır.

Durma noktası belirlemek için ilgili satır üzerinde F9 tuşu kullanılmalı veya aynı işlem Debug | Windows | Breakpoints penceresi yardımıyla yapılmalıdır.

VS.NET hata ayıklayıcısı aktif iken değişken değerlerinin izlenmesi amacıyla izleme (Watch) penceresi kullanılabileceği gibi ilgili değişken üzerinde sağ tıklayıp hızlı izleme (Quick Watch) penceresi ile değer görüntülemesi veya düzenlemesi yapılabilir.

Ekran Çıktısı 23, VS.NET Hızlı İzleme Penceresi

F10 ve F11 tuşları sırasıyla bir satırı işletmek ve satır içerisinde bulunan rutinlere giriş için kullanılmalıdır.

Aşağıda aynı işlevi yapan ancak farklı kullanıcı ara yüzlerine sahip uygulamalar gösterilmiştir.

Ekran Çıktısı 24, Farklı Uygulama Türleri

Bir insan hangi limana ulaşmak istediğini biliyorsa, onun için her rüzgar uygundur.

Albert Einstein

Bölüm

4

.NET Programlama Temelleri

P

rogramlama kavramı özellikle son on yıldır önemli değişimler geçirmektedir. Programlama ile geçmişte öncelikli olarak ifade edilen programlama dili kullanma becerisi iken, günümüzde programlama, IDE kullanımı, alt yapı bilgisi, programlama dili özellikleri, kullanıma sunulan hazır yapılar hakkında bilgi sahibi olmak gibi çok daha geniş yelpazedeki kavramların bütünü ile ifade edilebilmektedir.

.NET mimarisi üzerinde kaliteli programlama yapıyor olmak da yukarıda aktarılan temel kavramlar konusunda bilgi sahibi olmayı gerektirmektedir.

Bu bölümde olabildiğince detaylandırılmış şekilde .net programlama temelleri konusunda bilgiler sunulacaktır.

Teorik bilgilerin ağırlıklı olarak üzerinde durulacağı bu bölümün tüm okuyucular tarafından okunması önerilmektedir.

Kavramlar

Kitabın daha önceki bölümlerinde verilen bilgiler ışığında ve .NET programlama çerçevesinde temel kavramlar hakkında bu konu başlığında bilgiler sunulacaktır.

Nesne Tabanlı Mimari

.NET programlama tamamen nesne tabanlıdır. Bir başka anlatımla, kullanılan tüm hazır ve geliştirilen yapılar nesne tabanlı programlama standartlarındadır.

NTP, tasarım ve geliştirme süreçleri ile birlikte kitabın bir sonraki bölümünde tüm detaylarıyla aktarılacaktır.

NTP hakkında kavramsal düzeyde bilgisi olmayan okuyucuların öncelikle kitabın bir sonraki bölümü başlangıç konularını kısaca incelemeleri önerilmektedir.Bu bölümde tam olarak anlaşılmayan bazı konu ve kavramlar daha sonraki bölümünün tamamlanması ile daha iyi anlaşılacaktır.

Örneğin C# dilinde int tanımlaması ile tamsayı tipi ifade edilmektedir. Rezerve bir kelime olan int, farklı metot ve özellikleri içeren nesne tabanlı mimari standartlarında tanımlanmıştır.

Sınıf (Class) ile Yapı (Structure) birbirinden farklıdır. int gibi amacı sadece değer saklamak olan tipler yapı tipindedir. Kitabın ilerleyen bölümlerinde bu konu hakkında daha detaylı bilgi sunulacaktır.

VS.NET ile kodlama sırasında otomatik kod tamamlama özelliği yardımıyla "int." yazıp Ctrl + Space kombinasyonu ile int tipi üyelerine ulaşmak mümkündür.

Assembly

İçerisinde IL kodu ve Metadata bilgisi içeren kavramsal yapı Assembly olarak ifade edilebilir. Assembly içerisindeki bilgiler PE yapısına sahip bir dosyada saklanmaktadır. Assembly tanımını, daha üst seviyede, içerisinde sınıf ve nesne tanımları ile kod gerçekleştirimlerini içeren yapı olarak yapmak mümkündür.

Ad Alanları

İngilizce karşılığıyla Namespace olarak ifade edilebilecek ad alanları mantıksal ve soyut kapsayıcılar olarak tanımlanabilir.

**using** System;

**namespace** DotNetKitabı.ÖrnekUygulamalar {

**class** HesapMakinesi {

**static** **string** Hesapla(**int** x) {

Kod 12, Örnek C# Kaynak Kod Parçası

Yukarıdaki kod parçası incelendiğinde, DotnetKitabı.ÖrnekUygulamalar ad alanı içerisinde bir sınıf tanımı yapıldığı görülebilir. Aynı Assembly içerisinde benzer şekilde aşağıdaki sınıf tanımı da rahatlıkla yapılabilir.

**using** System;

**namespace** Sistem.Matematik {

**class** HesapMakinesi {

**static** **string** Hesapla(**int** x) {

Kod 13, Örnek C# Kaynak kod parçası

İsimlerinin aynı olmasına rağmen farklı ad alanları içerisinde olan HesapMakinesi sınıfları aşağıdaki şekilde kullanılabilir.

Console.Write(Sistem.Matematik.HesapMakinesi.Hesapla(12));

// veya

Console.Write(

DotnetKitabı.ÖrnekUygulamalar.HesapMakinesi.Hesapla(12));

// veya

**using** Sistem.Matematik;  
...  
Console.Write(HesapMakinesi.Hesapla(12));

Kod 14, Örnek kullanım

Ad alanları ile, sınıfların bütünleşik isim nitelendirmesi (ing. Fully Qualified Names) avantajına ek olarak ortak bir isim şemsiyesi altında birbirleri ile mantıksal olarak ilgili sınıfların kategorize edilebilmesi de hedeflenmektedir.

Assembly dosya adları ile ad alanları birbirleri ile karıştırılan kavramlardır. Örneğin Assembly dosya adı DotnetKitabı.Ortakİşlemler.dll olduğu değerlendirildiğinde, bu Assembly içerisinde birden çok ad alanı olabilir. Ad alanlarının fiziksel dosya adı ile herhangi bir ilişkisi bulunmamaktadır.

VS.NET ile proje özellikleri penceresinden ön tanımlı (ing. Default) ad alan tanımlaması yapılabilir.

Kullanım Detayları

Verilen örnek kod parçalarından da anlaşılabileceği gibi bir sınıfı kullanmadan önce ilgili sınıfın hangi ad alanı içerisinde olduğu derleyici tarafından çözümlenmelidir.

Bu çözümleme, sınıfın bütünleşik isim nitelendirmesi ile, DotnetKitabı.ÖrnekUygulamalar.HesapMakinesi şeklinde olabileceği gibi, C# için using, VB.NET için imports rezerve kelimeleri kullanarak da yapılabilir.

İç İçe Ad Alanı Kullanımı

Ad alan tanımlarını iç içe kullanmak mümkündür. Böylece aynı ad alanı içerisinde bir veya daha fazla gruplamalar oluşturulabilir.

| C# | VB.NET |
| --- | --- |
| **namespace** Test {  **public** **class** ÜstSınıf {  }  **namespace** Örnekler {  **public** **class** AltSınıf {  }  }  } | **Namespace** Test  **Public** **Class** ÜstSınıf  **End** **Class**  **Namespace** Örnekler  **Public** **Class** AltSınıf  **End** **Class**  **End** **Namespace**  **End** **Namespace** |

Kod 15, İç İçe Ad Alanı Kullanımı

Rumuz Kullanımı

Ad alanları için rumuz kullanımı desteklenmektedir. Aşağıda bunu örnekleyen kod parçası gösterilmiştir.

C#

// System.Collections.Generic ad alanı için rumuz kullanılmaktadır.

**using** Generics = System.Collections.Generic;

// System.Console sınıfı için rumuz kullanılmaktadır.

**using** Konsol = System.Console;

**namespace** TestNS {

**class** TestClass {

**void** TestMethod(){

Generics::Dictionary ...;

Konsol.WriteLine("deneme");

}

}

}

VB.NET

' System.Collections.Generic ad alanı için rumuz kullanılmaktadır.

**Imports** Generics = System.Collections.Generic

' System.Console sınıfı için rumuz kullanılmaktadır.

**Imports** Konsol = System.Console

**Namespace** TestNS

**Class** TestClass

**Sub** TestMethod()

Generics.Dictionary ...

Konsol.WriteLine("deneme")

**End** **Sub**

**End** **Class**

**End** **Namespace**

Kod 16, Ad Alanları Rumuz Kullanımı

global rezerve kelimesi, global ad alanına referans vermek için kullanılmalıdır. Örneğin global::System.Console gibi.

Ortak Ata Sınıf

Tüm .NET sınıflarının atası System.Object sınıfıdır. Bir sınıf tanımlamasında belirtilsin veya belirtilmesin tüm sınıflar System.Object sınıfını otomatik olarak miras almaktadır.

Ortak Tip Sistemi - CTS

Ortak dil belirtimi (CLS) yapısının bir parçası olan CTS, uygulamalar hangi programlama dili ile geliştirilirse geliştirilsin .NET uygulamalarında kullanılan tiplerin birbirleri ile uyumlu olmalarını sağlayan sistemdir.

CTS, farklı programlama dillerinde aşağıda listelenmiş olan tipler için ortak tanımlamaları içermektedir.

* Sınıf tipleri (Class types)
* Arabirim tipleri (Interface types)
* Yapı tipleri (Structure types)
* Tip üyeleri (Type members)
* Sayılabilir tipler (Enumeration types)
* Delege tipler (Delegate types)
* Temel tipler (Intrinsic types, Primitive Types)

Programlama dili yapısından bağımsız olarak, IL kod içerisinde aynı anlamı ifade eden yukarıda listelenmiş olan CTS tipleri kullanım detayları kitabın ilerleyen bölümlerinde sunulacaktır.

Aşağıda ortak tip sistemi içerisinde tanımlı temel tipler, tanımlamaları ve açıklamaları ile gösterilmiştir.

| Tip | Açıklama |
| --- | --- |
| System.Byte | 0 - 255 arası tamsayı değerler |
| System.SByte | -128 – 127 arası tamsayı değerler |
| System.Int16 | -32768 – 32767 arası tamsayı değerler |
| System.Int32 | –2,147,483,647 – 2,147,483,647 arası tamsayı değerler |
| System.Int64 | -9223372036854775808 – 9223372036854775807 arası tamsayı değerler |
| System.UInt16 | 0 – 65535 arası tamsayı değerler |
| System.UInt32 | 0 – 4294967295 arası tam sayı değerler |
| System.UInt64 | 0 – 18446744073709551615 arası tam sayı değerler |
| System.Single | 7 basamak hassasiyetli ondalık sayısal değerler |
| System.Double | 16 basamak hassasiyetli ondalık sayısal değerler |
| System.Object | Tüm nesneler için ata sınıf |
| System.Char | Unicode karakter |
| System.String | Karakter dizi değerler |
| System.Decimal | 28 basamak hassasiyetli ondalık sayısal değerler |
| System.Boolean | Sadece True ve False değeri alabilen tip |

Tablo 8, CTS Temel Tip Tanımları ve Açıklamaları

Farklı dillerde farklı kelimeler ile tanımlama yapılabilmesine rağmen, yapılan tanımlamalar ortak tip sistemi içerisinde aynı tipi ifade etmektedir.

// Aşağıdaki tanımlamalar birebir aynıdır.

**int** x = 12;   
System.Int32 x = 12;

Kod 17, Farklı Şekilde Tip Tanımlamaları

Temel tiplerin çalışma zamanında farklı donanımsal mimarilerde de ortak olduğu unutulmamalıdır. 32Bit mimariye sahip bir sistem ile 64Bit mimariye sahip bir sistemde çalışan .NET kodunda tüm tipler aynıdır.

Aşağıdaki tabloda temel tiplerin farklı programlama dillerinde alternatif tanımlanma şekilleri gösterilmiştir.

| Tip | C# | VB.NET | Delphi.Net |
| --- | --- | --- | --- |
| System.Byte | byte | Byte | Byte |
| System.SByte | sbyte | Sbyte | ShortInt |
| System.Int16 | short | Short | SmallInt |
| System.Int32 | int | Integer | Integer |
| System.Int64 | long | Long | Int64 |
| System.UInt16 | ushort | UShort | Word |
| System.UInt32 | uint | UInteger | Cardinal |
| System.UInt64 | ulong | ULong | UInt64 |
| System.Single | float | Single | Single |
| System.Double | double | Double | Double |
| System.Object | object | Object | Object |
| System.Char | char | Char | Char |
| System.String | string | String | String |
| System.Decimal | decimal | Decimal | Decimal |
| System.Boolean | bool | Boolean | Boolean |

Tablo 9, CTS Tipleri ve Farklı Dillerde Tanımlamaları

Int32 ve UInt32 tipleri çalışma zamanında optimize edilmektedir. Dolayısıyla sayaçlarda ve sık değeri değiştirilen tamsayı işlemlerinde bu tiplerin kullanımı önerilmektedir.  
  
Ondalık sayı işlemlerinde System.Double kullanımı önerilmektedir. System.Double tipi ile yapılan işlemler donanımsal olarak optimize edilmektedir.

Değer ve Referans Tipler

CTS, iki ana tip sistemi üzerine kuruludur.

* Değer Tipler
* Referans Tipler

Değer tipler, klasik programlama dilleri (C, C++, Pascal) ile kullanılan temel tipler (Tam sayılar, ondalık sayılar, karakterler vb.), yapı tipleri ve sayılabilir tipleri içermektedir. Değer tiplerin temel amacı ve kullanımı, adından da anlaşılabileceği gibi, belirli bir değeri saklamak ve gerektiğinde en hızlı şekilde sakladığı değeri döndürmektir.

Referans tipler, nesne yönelimli programlama çerçevesinde, nesneler, arabirimler, sınıflar gibi temel amacı işlevsellik sağlamak olan tiplerdir. Tanımın referans olarak yapılmış olması bu tiplerde olan değişkenlerin gerçekte bir başka hafıza adresine referans veriyor olmasından kaynaklanmaktadır.

Tamamıyla nesne tabanlı olan .NET içerisinde, kitabın ilerleyen bölümünde de detaylarıyla açıklanacağı üzere, kullanılan her yapı belirtilsin veya belirtilmesin System.Object sınıfını ata sınıf olarak kabul etmektedir. Bu çerçevede değerlendirildiğinde, değer tipleri de birer nesne olarak düşünülebilir. Aynı şekilde referans tipler de birer nesnedir.

Değer tipleri ile referans tipleri ayrımı System.ValueType sınıfında gizlidir. Değer tipleri bu sınıfı miras almaktadır. Aşağıda sınıf hiyerarşik yapısı gösterilmiştir.

Şekil 7, Değer ve Referans Tipleri Hiyerarşik Yapısı

Değer tipler stack üzerinde, referans tipler heap üzerinde saklanmaktadır.

Klasik programlama ile karşılaştırıldığında, referans tipleri işaretçi tipler (ing. Pointer Type) benzeri düşünmek ve değerlendirmek yanlış olmayacaktır. Referans tipler, işaretçi tiplerde olduğu gibi null değere eşitlenebilmekte, yine benzer şekilde hafıza ayrımı yapıldıktan sonra kullanılabilmektedir.

Aşağıda gösterilen Ansi C ve VB.NET kaynak kodlarındaki benzerliğe dikkat ediniz.

**int** \*i = **null**;  
i = (**int**)malloc(**sizeof**(int));

Kod 18, İşaretçi Tip Örnek C Kodu

**Dim** o **As** **Object** = **nothing**  
**o = New Object**()

Kod 19, Referans Tip Örnek VB.NET Kodu

Yukarıdaki kod örneklerinin her ikisinde de arka planda, temelde, hafıza ayrımı yapılmaktadır. Bu benzerlik tüm nesne tabanlı platform ve programlama dilleri için geçerlidir.

.NET 2.0 sürümünden itibaren değersizleşebilen tip kullanımı mümkündür. Buna rağmen, Değersizleşebilen tipler ile değer tiplerine null değer ataması yapılabilmektedir. Değer tipe null ataması yapılması ile referans tipler birbirleri ile karıştırılmamalıdır.  
  
CLR, işaretçi tip kullanımına ve daha alt seviye işlemlere limitli destek sağlamaktadır ve bu yapıların kullanımı güvensiz (ing. UnSafe) olarak nitelenmektedir.

Programlama Dil Yapıları

.NET, kitabın daha önceki konularında da detaylarıyla aktarıldığı üzere dil bağımsızlığı sağlamaktadır. Dil bağımsızlığını sağlayan CLS, aynı zamanda programlama yapmak için tercih edilen dilin önemini de önemli ölçüde azaltmaktadır.

Günümüzde .NET çatısı altında programlama için, bu alt yapıya destek veren programlama dili sayısı oldukça yüksektir. Farklı cümle dizilişleri ve dil yapıları ile programlama dilleri birbirlerinden ayrılmalarına rağmen, farklı derleyiciler tarafından üretilen kodun standartları tamamen aynıdır.

.NET bir bütün olarak değerlendirildiğinde, programcıların tercih ettikleri programlama dilinin teorik olarak herhangi bir bağlayıcılığı, farklı programlama dillerinin birbirlerine herhangi bir üstünlüğü bulunmamaktadır.

Programlama dili seçiminde, programcının dil ve cümle yapısına alışkanlığı ve geçmiş deneyimi gibi kavramların değerlendirilmesi önerilmektedir.

Kitabın bu bölümünde, anlatımda herhangi bir programlama dili baz alınmak yerine, farklı diziliş, cümle yapısı ve işlevlerin örneklemesi yapılacaktır. Aynı zamanda önemli görülen bazı C#, VB.NET ve Delphi dil yapıları ile ilgili bilgiler sunulacaktır.

Programlama Dil Grameri

Her programlama dili, derleyici adımlarına benzer şekilde, sözcüksel (ing. Lexical) ve sözcük dizim (ing. Syntax) kurallarından oluşmaktadır.

Sözcüksel kurallar kavramı ile ifade edilmek istenen, karakterlerin birleşerek ne şekilde kelimeler oluşturabildiğini tanımlamak ve somut olarak ortaya koymaktır. Sözcük dizim kuralları, kelimelerin hangi kurallar ile birleşerek programlama dilinin tamamını oluşturduğunu tanımlayan kavramdır.

Yukarıda anlatılan kuralların formel olarak tanımlanabilmesi için standart olarak benimsenmiş farklı gramerler kullanılmaktadır. Bu standartlardan en çok kullanılan ve LEX ve YACC gibi yazılımlar tarafından da desteklenen BNF (Backus–Naur Form) standardıdır.

Aşağıda örnek BNF grameri gösterilmiştir.

S := '-' FN |

FN

FN := DL |

DL '.' DL

DL := D |

D DL

D := '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'

Kod 20, Örnek BNF Notasyonu

Yukarıdaki gramer basit bir programlama dilini tanımlamaktadır. Tanımlanan programlama dili, sadece pozitif ve negatif olabilen tam sayı ve ondalıklı sayıların dizilişini desteklemektedir.

Günümüzde kullanılan popüler programlama dillerinin de yukarıda basitçe örneklenen mimariyi baz aldığı ve programlama dillerinin daha geniş ve detaylı bir gramer yapısı olduğu unutulmamalıdır.

Temel Özellikler ve Kavramlar

Bu konu başlığında, başlangıç seviyesi niteliğinde temel dil özellikleri, sözcüksel ve sözcük dizim kuralları hakkında bilgiler verilecektir.

Yazılım Temelleri

Yapı ve mimarisi ne olursa olsun her yazılım üç adet temel kavramdan oluşur. Bir başka anlatımla, her yazılım üç nitelik üzerine kuruludur.

* Sırayla İşletim (Sequential Implementation)
* Dallara Ayrılma (Branching)
* Döngüsel Yapılar (Repeatative Structures)

Sırayla işletim, çalıştırılabilir kod dosyasında bulunan komutların başlangıç noktasından itibaren sırayla çalıştırılmasını ifade etmektedir. Döngüsel yapılar, bazı komut veya komut setlerinin sonlu sayıda bir döngü içerisinde çalıştırılmasını, dallara ayrılma ise belirli komut veya komut setlerinin yapılan bir karşılaştırma sonucuna göre çalıştırılacağı veya çalıştırılmayacağını ifade etmektedir.

Kaynak Kod Dosyaları

Kaynak kod dosyaları, herhangi bir metin editörü ile oluşturulabilecek metin bazlı dosyalardır. Kaynak kod dosya uzantıları C# için .cs, VB.NET için .vb, Delphi.NET için .pas şeklindedir.

Belirteçler

Belirteçler (ing. Identifiers), metinsel sözcüklerdir. Değişken adları, tipler, ad alan isimleri, paket isimleri belirteç örnekleridir.

Belirteç isimleri \_ (Alt çizgi) veya alfabetik karakterle başlayabilir ve içerisinde sadece alfa nümerik karakterler içerebilirler.

Delphi.NET ve VB.NET belirteç isimlerinde büyük – küçük harf ayrımı yapmamaktadır. C# belirteçleri büyük – küçük harf duyarlıdır.

İfadeler ve Bloklar

İfadeler (ing. Statements), mantıksal olarak gruplandırılmış yazılım parçaları olarak tanımlanabilir.

Bloklar, temel olarak bir karşılaştırma işlemi sonucu çalıştırılacak veya döngüler için kullanılacak kod grubunu belirtmek için kullanılabilir.

Örneğin

**if** (a == b) {

// karşılaştırma işlemi sonucu çalıştırılacak ifadeler bloğu

}

Kod 21, Örnek C# Bloğu

**If** a = b **Then**

' karşılaştırma işlemi sonucu çalıştırılacak blok

**End If**' veya  
  
**Select** **Case** x

**Case** 1

...

**Case** 2

...

**End Select**

Kod 22, Örnek VB.NET Bloğu

**If** a = b **Then**

**Begin**

// karşılaştırma işlemi sonucu çalıştırılacak blok

**End**

Kod 23, Örnek Delphi.NET Bloğu

Yukarıdaki örneklerden de görüleceği üzere, C# için { } karakterleri, Delphi.NET için Begin End kelimeleri blok tanımlamalarında kullanılmaktadır.

VB.NET blok tanımı, bloğun amacı doğrultusunda yapılan tanım ve genelde End sözcüğü ile başlayan blok sonu ile yapılmaktadır.  
  
VB.NET blok tanımının bu şekilde olmasını en önemli sebebi diğer programlama dillerinde ifade ayıracının (; karakteri) VB.NET için tek bir satır olmasıdır. Bir başka anlatımla VB.NET için temel ifade bir satırdır.   
  
Delphi ve C# için blok kuralları baz alınmasına rağmen VB.NET için bu kural satır bazlıdır. Örneğin aşağıdaki C# kodu geçerli olmasına rağmen  
  
if (a ==  
 b)   
{}  
  
VB.NET için benzer bir kullanım  
  
If a =  
b  
Then  
End If  
  
diziliş doğru olmasına rağmen geçersizdir. Birden fazla satır kullanımı VB.NET için \_ (Alt çizgi) karakteri ile sağlanabilir. Örneğin   
  
If a \_  
 = b \_  
Then  
End If

Bloklar, aynı zamanda kapsamın (ing. Scope) belirlenmesi amacıyla da kullanılabilir. Aşağıdaki örnekte iç içe iki blok ve kapsam gösterilmiştir.

**class** Program {

**static** **void** Main(**string**[] args) {

**int** z = 12; **{**

// j isimli değişken sadece bu blok içerisinde geçerlidir.

**int** j = 12;

**}**

Console.WriteLine(z.ToString());

Console.WriteLine(j.ToString()); // Hatalı kullanım,   
 // j geçersizdir.

}

}

Kod 24, *Blok ve Kapsam*

Yukarıdaki örnekte, j değişkeni sadece tanımlandığı blok içerisinde geçerlidir.

Ek Açıklamalar

Kaynak kod içerisine bilgi eklemek amacıyla kullanılan ve derleyiciler tarafından dikkate alınmayan ek açıklamalar (ing. Comments), kaynak kod hakkında bilgi vermek amacıyla kullanılan yapılardır.

Aşağıda farklı programlama dillerinde ek açıklama tanımlarının ne şekilde yapılabileceği gösterilmiştir.

| C# | VB.NET | Delphi.Net |
| --- | --- | --- |
| // Tek satır ... /\* Çoklu satır  Ek açıklama  örneği \*/ | ' Tek satır açıklama  **Rem** tek satır ... | // Tek satır ... (\* Çoklu satır  Ek açıklama  örneği \*) |

Kod 25, Ek Açıklama Tanımlamaları

Ek açıklamalar, kaynak kodun toplam kalitesini ve okunulurluğunu artıran yapılardır. Özellikle anlaşılması karmaşık kaynak kod parçalarında kullanımı önerilmektedir.

Rezerve Edilmiş Kelimeler

Rezerve edilmiş kelimeler (ing. Reserved Words), belirteç olarak kullanılamayacak ve programlama dili gramerinde özel anlam ifade eden kelimelerdir.

C# ve Delphi için rezerve edilmiş kelimeler belirteç olarak kullanılamamakla birlikte, VB.NET bu kullanıma izin vermektedir. Örneğin Sub VB.NET için rezerve edilmiş bir kelime olmasına rağmen [] parantezleri içerisinde kullanıldığında belirteç olabilmektedir.  
  
Dim [Sub] as Integer  
[Sub] = 12  
  
Geçerli bir kullanımdır.

Farklı programlama dilleri için hangi kelimelerin rezerve edilmiş oldukları hakkında bilgi programlama dili dokümantasyonundan elde edilebilir.

Anlatımlar

Anlatımlar (ing. Expressions), değerler, değişkenler, operatörler veya fonksiyon çağrımlarından oluşan değer döndüren yapılardır.

Örneğin

1

5 \* 3

(a + 12) / 5

"deneme" + "1 2 3"

anlatımlara örnek verilebilir. Anlatım sonucu mutlaka bir değer döndüğüne dikkat edilmelidir.

Programlama Dil Yapıları

Temel Ata Sınıf

.NET ile çalışmak ileri seviye nesne tabanlı programlama bilgisini zorunlu kılmaktadır. .NET kapsamında kullanılan kısaca her şeyin ata sınıfı System.Object sınıfıdır.

Klasik programlama dillerinde kullanılan temel tipler (ing. Primitive Types) de dahil olmak üzere tipler, değerler, değişkenler, diziler, sabitler vb. System.Object sınıfını ata sınıf olarak kabul etmektedir.

Bu kapsamda aşağıdaki kullanımlar tamamen geçerlidir.

| C# | VB.NET |
| --- | --- |
| **string** s = 12.ToString();  **int** i = "Deneme".Length;  **double** [] dizi =  **new** **double**[3] {1.1, 1.2, 1.3};  **int** diziBoyutu = dizi.Length; | **Dim** s **As String** = 12.ToString()  **Dim** i **As Integer** = "Deneme".Length  **Dim** dizi() **As Double**  dizi = **New Double**(2) {1.1, 1.2, 1.3}  **Dim** diziBoyutu **As Integer**  diziBoyutu = dizi.Length |

Kod 26, Nesne Tabanlı Mimari Örnekleri

Nesne tabanlı programlama ve System.Object sınıfı hakkında detaylı bilgi kitabın ilerleyen bölümlerinde sunulacaktır.

Değersizlik

Değersizlik, daha önce işaretçi tip işlemlerinde sık kullanılan null, nothing veya nil olarak ifade edilebilen kavramdır.

Değersizlik, değişken içeriğinin boş olduğunu, değişkenin herhangi bir değer içermediğini belirtmektedir. Kitabın ilerleyen bölümlerinde değersizlik hakkında detaylı bilgiler sunulacaktır.

Sözcüksel İfadeler

Aşağıda temel sözcüksel ifadeler ve örnekleri gösterilmiştir.

| Sözcüksel İfade | C# | VB.NET | Delphi.Net |
| --- | --- | --- | --- |
| Tam sayı | Onluk Sistem:  12 Onaltılık Sistem: 0xA | Onluk Sistem: 12 12 Onaltılık Sistem: &HA | Onluk Sistem: 12 12 Onaltılık Sistem: $A |
| Ondalık Sayı | 3.14 | 3.14 | 3.14 |
| Karakter Dizi (String) | "Deneme" | "Deneme" | 'Deneme' |
| Karakter | 'A' | "A"c | 'A' |
| Mantıksal | true / false | True / False | True / False |
| Değersizlik | null | Nothing | Nil |

Tablo 10, Temel Sözcüksel İfadeler

C#, \ karakterini kaçış karakteri (ing. Escape Character) olarak tanımlamaktadır. Örneğin "Ali\" ve \"Veli" şeklinde kullanım "" karakterlerini de karakter dizi içerisine dahil etmektedir.

Benzer şekilde, "C:\\Documents And Settings\\Users" karakter dizi değeri C:\Documents And Settings\Users değerini ifade etmektedir. Karakter dizi bir değer içerisindeki kaçış karakterinin göz ardı edilmesi için @ operatörü kullanılabilir. @"C:\Documents And Settings\Users" şeklindeki kullanım birebir "C:\\Documents And Settings\\Users" kullanımı ile aynıdır.

Kaçış karakteri aynı zamanda C# tarafından ön tanımlı bazı değerleri ifade etmek için kullanılabilir. Aşağıdaki tabloda kaçış karakteri hakkında detaylı bilgi sunulmuştur.

| Kullanım | Açıklama |
| --- | --- |
| \' | Karakter sözcüklerde ' karakteri kaçışı |
| \" | Karakter dizi sözcüklerde " kaçışı |
| \\ | \ karakteri için kaçış |
| \0 | 0 karakteri |
| \a | Alarm (Karakter 7) |
| \b | Geri karakteri (ing. Backspace) (Karakter 8) |
| \f | Form besleme (Karakter 12) |
| \n | Yeni satır (Karakter 10) |
| \r | CR karakteri (Karakter 13) |
| \t | Yatay tab (Karakter 9) |
| \v | Dikey alıntı (Karakter 11) |
| \xn | Onaltılık sisteme göre n karakter kodunu belirtir. |

Tablo 11, C# Kaçış Karakteri Kullanımları

VB.NET ve Delphi.NET için sırasıyla " ve ' karakterlerinin karakter dizi değer içerisinde yer alabilmesi için iki kez yazılması gereklidir. Örneğin 'Ali''nin' gibi.

Delphi.NET ile # karakteri kullanılarak karakter kodu kullanımı mümkündür. Örneğin #65 birebir 'A' karakter değerine eşittir.

VB.NET, tarih sözcüksel ifadelerini desteklemektedir. # # karakterleri arasına yazılan tarih geçerlidir. Örneğin #01/01/2002 02:00 PM# geçerli bir tarih değerini ifade etmektedir.

Değişkenler

Değişkenler, hafıza adresleri için rumuzlar olarak tanımlanabilir. Aşağıda değişken tanımlama ve ilk değer atama ile ilgili örnekler listelenmiştir.

| C# | VB.NET |
| --- | --- |
| // Değişken tanımlama  **int** x;  // Değişken tanımlama ve ilk  // değerini atama **int** y = 12;  // Birden fazla değişken tanımlama  **int** z, m = 12;  // Değişken tanımlama ve ilk  // değerini atama **int** k = z + m; | ' Değişken tanımlama  **Dim** x **As Integer**  ' Değişken tanımlama ve ilk  ' değerini atama **Dim** y **As Integer** = 12  ' Birden fazla değişken tanımlama  **Dim** z **As Integer**, m **As Integer** = 12  ' Değişken tanımlama ve ilk  ' değerini atama **Dim** k **As** **Integer** = z + m |

Kod 27, Değişken Tanımlama

Değişken tanımlaması için özel bir tanımlama bloğu gerekmemektedir. Aşağıdaki kaynak kod örneği geçerlidir.

**Dim** j **As** **Integer** = 12

// Konsol ekranına yazdır.

Console.WriteLine(j)

' Yeni değişken tanımla.

**Dim** s **As** **String** = "xyz"

Kod 28, Örnek VB.NET Değişken Tanımlaması

Değişken tiplerinin, değişken tanımlama aşamasında belirleniyor olması C# 3.0 ve VB.NET 9.0 sürümünden (.NET 3.0) itibaren zorunlu değildir.

VB.NET, VS.NET proje özellikleri penceresi ile tanımı düzenlenebilecek veya kaynak kod içerisinde kullanılabilecek derleyici komutu Option Explicit ile değişken tanımlama kurallarını yönetmektedir. Option Explicit On, değişkenlerin kullanılmadan önce tanımlı olması gerektiğini, Option Explicit Off, tanımlama yapılmadan da değişken kullanımına izin vermektedir.  
  
Kullanılan değişkenlerin tipleri ile birlikte tanımlanması veya değişken tiplerinin derleme anında belirlenebilir olmaları önerilmektedir. Değişkenlerin gerek kendileri gerek tiplerinin tanımlanmadan kullanımı önemli programlama hatalarına yol açabilmektedir.

Yerel değişken tiplerinin dolaylı olarak tanımlanması (ing. Implicitly Typed Local Variables), derleme anında değişken tipinin, değişkene atanan değer ile belirlenmesini ifade etmektedir. Aşağıda tanımlama örnekleri gösterilmiştir.

| C# | VB.NET |
| --- | --- |
| // x değişkeninin tipi belirtilmese // dahi ilk değer atama ile derleme // zamanında tipi  // belirlenebilmektedir.  **var** x = 12; **var** y = 12.3; **var** s = "deneme"; **var** dizi = **new int** [] {1, 2, 3}; | ' x değişkeninin tipi belirtilmese ' dahi ilk değer atama ile derleme ' zamanında tipi  ' belirlenebilmektedir.  **Dim** x = 12 **Dim** y = 12.3 **Dim** s = "deneme" **Dim** dizi = {1, 2, 3} |

Kod 29, Dolaylı Yerel Değişken Tipleri

Yukarıdaki tanımlamalar birebir aşağıdakiler ile aynıdır.

| C# | VB.NET |
| --- | --- |
| **int** x = 12; **double** y = 12.3; **string** s = "deneme"; **int []** dizi = **new int** [] {1, 2, 3}; | **Dim** x **As Integer** = 12 **Dim** y **As Double** = 12.3 **Dim** s **As String** = "deneme" **Dim** dizi() **As Integer** = {1, 2, 3} |

Kod 30, Direkt Yerel Değişken Tipleri

Derleme anında değişken tiplerinin belirlenebilmesi için, tanımlama aşamasında değişkene atanan değerden tip çözümlenebilmesinin yapılabilir olması gerekmektedir.

Aşağıdaki kullanımlar geçersizdir.

**var** x;

**var** x = **null**;

// x isimli değişken tipi derleme zamanında belirlenemez.

Kod 31, Geçersiz Dolaylı Tip Tanımları

Diziler

Diziler (ing. Arrays), tipleri birebir aynı olan ve içeriklerine sayısal bir indeks aracılığı ile ulaşılabilen değişkenler listesi olarak tanımlanabilir. Diziler tek boyutlu veya çok boyutlu olabilirler. .NET aynı zamanda çok boyutlu diziler için sabit (ing. Rectangular) ve değişken (ing. Jagged) çok boyutlu dizi tiplerini desteklemektedir.

Aşağıda temel dizi tanımlamaları örneklenmiştir.

C#

// Tek boyutlu diziler

**int**[] dizi1 = **new** **int**[5];

**int**[] dizi2 = { 1, 2, 3, 4, 5 };

// Düzgün çok boyutlu diziler

**int**[,] dizi3 = **new** **int**[2, 2];

**int**[,] dizi4 = { {1, 2}, {3, 4} };

**int**[,,] dizi5 = **new** **int**[5, 3, 5];

// Değişken çok boyutlu diziler

// Dizi içerisinde dizi

**int**[][] dizi6;

// Dizi içerisinde dizi içerisinde dizi

**int**[][][] dizi7;

VB.NET

' Tek boyutlu diziler

**Dim** dizi **As** **Integer**()

dizi = **New** **Integer**(4) {} 'veya Dim dizi(4) as Integer

**Dim** dizi2 **As** **Integer**() = {1, 2, 3, 4, 5}

' Düzgün çok boyutlu diziler

**Dim** dizi3(,) **As** **Integer**

dizi3 = **New** **Integer**(1, 1) {}

**Dim** dizi4(,) **As** **Integer** = {{1, 2}, {3, 4}}

**Dim** dizi5(,,) **As** **Integer**

dizi5 = **New** **Integer**(4, 2, 4) {}

' Değişken çok boyutlu diziler

' Dizi içerisinde dizi

**Dim** dizi6 **As** **Integer**()()

' Dizi içerisinde dizi içerisinde dizi

**Dim** dizi7 **As** **Integer**()()()

Kod 32, Çeşitli Dizi Tanımlama Örnekleri

Dizi indeksleri her zaman 0'dan başlamaktadır ve dizi boyutu – 1 son değerine sahiptir. C# dizi boyut tanımı ile VB.NET birbirinden farklıdır.  
  
Yukarıdaki örnek kodlarda dizi1 isimli dizi boyutu 5'dir. C# ve VB.NET farkına dikkat ediniz.

**Tek Boyutlu Diziler**

Aşağıdaki örnek tek boyutlu dizi kullanımını göstermektedir.

C#

// dizi1 boyutu kadar dön.

**for** (**int** i = 0; i < dizi1.Length; i++)

dizi1[i] = i \* i;

// dizi1 içeriğini yazdır.

**for** (**int** i=0; i < dizi1.Length; i++)

Console.WriteLine("dizi1[{0}] = {1}", i, dizi1[i]);

VB.NET

' dizi1 boyutu kadar dön.

**For** i **As Integer** = 0 **To** dizi1.Length - 1

dizi1(i) = i \* i

**Next**

' dizi1 içeriğini yazdır.

**For** i **As Integer** = 0 **To** dizi1.Length - 1

Console.WriteLine("dizi1[{0}] = {1}", i, dizi1(i))

**Next**

Kod 33, Tek Boyutlu Dizi Kullanımı

Kod parçasının çıktısı aşağıdaki gibi olacaktır.

dizi1[0] = 0

dizi1[1] = 1

dizi1[2] = 4

dizi1[3] = 9

dizi1[4] = 16

**Çok Boyutlu Diziler**

Aşağıdaki örnekler çok boyutlu dizi kullanımını göstermektedir.

| C# | VB.NET |
| --- | --- |
| dizi3[0, 0] = 1;  dizi5[1, 1, 1] = 7;  dizi6 = **new** **int**[3][];  dizi6[0] = **new** **int**[] { 10, 11, 12 };  dizi6[1] = **new** **int**[] { 13 };  dizi6[2] = **new** **int**[5];  dizi6[2][0] = 1; | dizi3(0, 0) = 1  dizi5(1, 1, 1) = 7  dizi6 = **New** **Integer**(2)() {}  dizi6(0) = **New** **Integer**() {10, 11, 12}  dizi6(1) = **New** **Integer**() {13}  dizi6(2) = **New** **Integer**(4) {}  dizi6(2)(0) = 1 |

Tablo 12, Çok Boyutlu Dizi Kullanımı

Yukarıdaki kod örneğinde, sırasıyla iki boyutlu ve üç boyutlu olan dizi3 ve dizi5 isimli değişkenlere değer ataması yapılmaktadır.

Değer ataması ardından düzgün çok boyutlu olan dizi3 görünümü aşağıdaki gibi olmaktadır.

1, 0  
0, 0

Değişken çok boyutlu olan dizi6 değişkeni içeriği, tek boyutlu bir dizidir. Kod örneğinde, öncelikle dizi içeriği bir bütün olarak oluşturulmakta, daha sonra her dizi içeriği ayrı ayrı tek boyutlu dizi şeklinde ayrıca oluşturulmaktadır. Böylece dizi6 görünümü aşağıdaki şekilde olmaktadır.

10, 11, 12  
13  
1, 0, 0, 0, 0

**Dizi Metot ve Özellikleri**

System.Array sınıfı tüm dizilerin ata sınıfıdır. Bu sınıf içerisinde tanımlı olan metot ve özellikler tanımlanan tüm diziler için geçerlidir.

Aşağıda System.Array sınıfı temel üye bilgileri listelenmiştir.

| Üye Adı | Açıklama |
| --- | --- |
| Length | Dizi boyutunu döndürür. Çok boyutlu dizilerde dönen değer tüm dizinin toplam boyut bilgisidir. |
| Rank | Dizinin boyutunu döndürür. |
| IndexOf | Tek boyutlu dizilerde arama yapmak için kullanılmalıdır. |
| BinarySearch | Tek boyutlu sıralanmış dizilerde ikil arama yapmak için kullanılmalıdır. |
| Copy | Bir dizi içeriğini bir başka diziye kopyalamak için kullanılmalıdır. |
| Sort | Dizi içeriğini sıralamak için kullanılmalıdır. |

Tablo 13, System.Array Sınıfı Temel Üye Bilgileri

System.Array sınıfı özellik ve metotlarının kullanımı ile ilgili bazı örnekler aşağıda gösterilmiştir.

| C# | VB.NET |
| --- | --- |
| **int**[] y = **new** **int**[] { 6, 8, 3 };  **int**[] x = **new** **int**[y.Length];  // y isimli dizi içeriğini  // x isimli diziye kopyalar.  Array.Copy(y, x, y.Length);  // x isimli dizi içeriğini sıralar.  Array.Sort(x);  // Sıralanmış bir dizi içerisinde  // ikil arama yapar. Dizinin sıralı  // olması gerektiğine dikkat ediniz.  // Aşağıdaki kod {3, 6, 8} dizisi  // içerisinde ikil arama yapıp 2  // değerini döndürecektir.  Array.BinarySearch(x, 8);  // Herhangi bir dizi içerisinde  // dizi başlangıcından başlayarak  // arama yapar ve ilk bulduğu  // dizi elemanı indeksini döndürür.  // Aşağıdaki kod {6, 8, 3} dizisi  // içerisinde arama yapıp 1  // değerini döndürecektir.  Array.IndexOf(y, 8); | **Dim** y() **As** **Integer** = {6, 8, 3}  **Dim** x(2) **As** **Integer**  ' y isimli dizi içeriğini  ' x isimli diziye kopyalar.  Array.Copy(y, x, y.Length)  ' x isimli dizi içeriğini sıralar.  Array.Sort(x)  ' Sıralanmış bir dizi içerisinde  ' ikil arama yapar. Dizinin sıralı  ' olması gerektiğine dikkat ediniz.  ' Aşağıdaki kod {3, 6, 8} dizisi  ' içerisinde ikil arama yapıp 2  ' değerini döndürecektir.  Array.BinarySearch(x, 8)  ' Herhangi bir dizi içerisinde  ' dizi başlangıcından başlayarak  ' arama yapar ve ilk bulduğu  ' dizi elemanı indeksini döndürür.  ' Aşağıdaki kod {6, 8, 3} dizisi  ' içerisinde arama yapıp 1  ' değerini döndürecektir.  Array.IndexOf(y, 8) |

Kod 34, System.Array Kullanım Örnekleri

Dinamik dizi oluşturmak için System.Collections.ArrayList sınıfı kullanılabilir. Örneğin  
  
C#:  
 ArrayList list = new ArrayList();  
 list.Add(12);  
 list.Add(5);  
 int[] z = (int[])list.ToArray(typeof(int));  
  
VB.NET:  
 Dim list As ArrayList = New ArrayList  
 list.Add(12)  
 list.Add(5)  
 Dim z() As Integer = \_   
 CType(list.ToArray(GetType(System.Int32)), Integer)

Karakter Dizileri

Karakter dizileri (ing. String), programlama yaparken sık kullanılan metin değer bazlı tiplerdir.

Karakter dizi değerleri için büyük küçük harf ayrımı ve sıralama nitelikleri aktif kültür bilgisine (ing. Culture) göre farklılıklar göstermektedir. Özellikle Türkçe alfabenin I, ı ve İ, i karakterlerinin büyük küçük değerlerinin İngilizce alfabe ile örtüşmemesi önemli hatalara yol açabilmektedir.

Karakter dizileri salt okunurdur ve karakter dizilerin değerlerini değiştirir gibi gözüken bazı metotlar aslında yeni bir karakter dizi nesnesi döndürmektedirler.

Karakter dizi tipleri System.String sınıfını ata sınıf olarak kabul etmektedir. Karakter diziler ile ilgili temel üye bilgileri ve kullanım örnekleri listelenmiştir.

| Üye Adı | Açıklama |
| --- | --- |
| Empty | Boş karakter dizi değeridir. ("")  **if** (s == **string**.Empty)  // Aşağıdaki kod ile aynı işlemi ifade eder. **if** (s == "") |
| Length | Karakter dizisi uzunluğunu (içerisindeki karakter sayısını) döndürür. |
| Chars | Belirtilen indekste bulunan karakteri döndürür. Aşağıdaki her iki kullanım da geçerlidir.  **string** s = "Test Metin Değeri";  // 3. pozisyonda (0'dan başlayarak)  // bulunan karakter değeri almak.  **char** c = s.Chars[3];   // Yukarıdaki kod ile birebir aynı işlemi // ifade eder. **char** c = s[3]; |
| StartsWith / EndsWith | Karakter dizinin sırasıyla belirtilen metin dizi değeri ile başlayıp / bittiğini döndürür.  // Başlar kontrolü **if** (s.StartsWith("Tes"))  // Biter kontrolü **if** (s.EndsWith("eri")) // |
| Replace | Karakter dizi değeri içerisindeki bir alt karakter dizi değerinin veya bir karakterin bir başkası ile değiştirilmiş yeni bir karakter dizi döndürür.  // Test değerini Deneme ile değiştir.  **string** s2 = s.Replace("Test", "Deneme"); |
| IndexOf | Karakter dizi içerisinde bir başka karakter dizi değeri veya karakterin pozisyonunu döndürür. Belirtilen değer bulunamazsa -1 değeri döndürülür.  // Karakter dizi içerisinde arama yapar ve  // aşağıdaki örnekte 5 değerini döndürür. **int** i = s.IndexOf("etin");   // Aktif kültür Türkçe ise 5 değerini döndürür. **int** i = s.IndexOf("Etİn", 0, StringComparison.CurrentCultureIgnoreCase); |
| ToLower / ToUpper | Karakter dizi değeri sırasıyla küçük harf ve büyük harf dönüşümünü yapar. |
| SubString | Karakter dizi değerinden belirtilen parametrelere göre belirli bir alt değeri döndürür.  // 5. pozisyondan başlayarak kalan bölümü al. // Sonuç Metin Değeri olacaktır. **string** s2 = s.SubString(5)   // Sonuç Test olacaktır. **string** s2 = s.SubString(0, 4) |
| IsNullOrEmpty | İlgili karakter dizisi değerini boş ve atanmamışlık değerine göre kontrol eder.  **if** (**string**.IsNullOrEmpty(s))  // Aynı işlem aşağıdaki kod ile yapılabilir. **if** (s == null || s.Length == 0)  Karakter dizi değerlerinin boş değer kontrolünde bu metodun kullanılması önerilmektedir. |

Tablo 14, Karakter Dizileri Temel Üye Bilgileri

**Karakter Dizi Kullanım Önerileri**

Karakter diziler salt okunur değerler oldukları için üzerlerinde yapılan işlemler yeni bir karakter dizi nesnesi döndürmektedir. Özellikle sık yapılan değer değiştirme işlemleri performans açısından negatif etkiye sahiptir. Karakter dizilerinin içeriklerinin değiştirilmesi için System.Text.StringBuilder sınıfı kullanılmalıdır.

Aşağıda StringBuilder sınıfı kullanımı ile ilgili bir örnek kod parçası gösterilmiştir.

**Dim** sb **As** StringBuilder = **New** StringBuilder("Bu")

sb.Append(" bir test ")

sb.Append("değeridir")

sb.Insert(2, " gerçek ")

Console.WriteLine(sb.ToString())

Kod 35, StringBuilder Örnek Kod Parçası

Karakter diziler için ön tanımlı eşitlik ve sıralama işlemleri aktif kültür bilgisine göre değişmektedir. Özellikle kaynak kod içerisindeki sabitler ile karşılaştırma yapılması bazı Türkçe karakterlerden kaynaklı programlama hatalarını da beraberinde getirmektedir.

**Dim** s1 **As** **String** = "i"

**Dim** s2 **As** **String** = "İ"

**Dim** s3 **As** **String** = "I"

**Dim** s4 **As** **String** = "ı"

Console.WriteLine("Aktif Kültür: {0}", \_

Thread.CurrentThread.CurrentCulture.DisplayName)

**Dim** b1 **As** **Boolean** = **String**.Compare(s1, s2) = 0  
**Dim** b2 **As** **Boolean** = **String**.Compare(s1, s2, **True**) = 0  
**Dim** b3 **As** **Boolean** = **String**.Compare(s1, s3) = 0  
**Dim** b4 **As** **Boolean** = **String**.Compare(s1, s2,   
 StringComparison.InvariantCultureIgnoreCase) = 0  
**Dim** b5 **As** **Boolean** = **String**.Compare(s1, s3,   
 StringComparison.InvariantCultureIgnoreCase) = 0  
**Dim** b6 **As** **Boolean** = **String**.Compare(s1, s2, **True**,   
 CultureInfo.InvariantCulture) = 0

**Dim** b7 **As** **Boolean** = **String**.Compare(s4, s3, **True**) = 0  
**Dim** b8 **As** **Boolean** = **String**.Compare(s4, s3, **True**,   
 CultureInfo.InvariantCulture) = 0  
**Dim** b9 **As** **Boolean** = **String**.Compare(s1, s2,   
 StringComparison.CurrentCultureIgnoreCase) = 0

Console.WriteLine("B1: {0}", b1)

Console.WriteLine("B2: {0}", b2)

Console.WriteLine("B3: {0}", b3)

Console.WriteLine("B4: {0}", b4)

Console.WriteLine("B5: {0}", b5)

Console.WriteLine("B6: {0}", b6)

Console.WriteLine("B7: {0}", b7)

Console.WriteLine("B8: {0}", b8)

Console.WriteLine("B9: {0}", b9)

Kod 36, Karakter dizi Karşılaştırma Örnekleri

Yukarıdaki kod parçası çıktısı aşağıdadır.

Aktif Kültür: Turkish (Turkey)

B1: False

B2: True

B3: False

B4: False

B5: True

B6: False

B7: True

B8: False

B9: True

Aktif kültürün İngilizce olması durumunda aynı kod parçası çıktısı aşağıda gösterilmiştir.

Aktif Kültür: English (United States)

B1: False

B2: False

B3: False

B4: False

B5: True

B6: False

B7: False

B8: False

B9: False

Programlama hatalarının önüne geçilmesi için karşılaştırma işlemlerinde StringComparison.InvariantCulture değerinin kullanımı önerilmektedir.

Örneğin

**Dim** s **As** **String** = "Ali"

**If** s.ToLowerInvariant() = "ali" **Then**

Console.WriteLine("OK")

**End** **If**

**If** **String**.Compare(s, "ALI", StringComparison.InvariantCultureIgnoreCase) **Then**

Console.WriteLine("OK")

**End** **If**

Kod 37, Karakter dizi Karşılaştırmaları

Yukarıdaki kod örneği aktif kültür değerine bakılmaksızın her zaman aşağıdaki çıktıyı üretecektir ve eşitlik karşılaştırmalarında kullanılması gereken doğru yöntemdir.

OK

OK

String.Compare metodu karakter dizi değerlerinin karşılaştırması için kullanılması gereken metottur. Metot, değerler belirtilen kriterlere göre eşit ise 0, ilk parametre değeri büyükse 1, ikinci parametre değeri büyükse -1 döndürmektedir.

Listelenen kaynak kodlar \Bölüm4\Strings klasöründe bulunmaktadır.

Değersizleşebilen Tipler

Değersizleşebilen tipler (ing. Nullable Types), değer tiplerine null (VS.NET için nothing) ataması yapılabilirliğini mümkün kılmaktadır.

Bilindiği üzere değer tiplerin temel amacı içerdiği tipe göre belirli bir değeri saklamak ve gerektiğinde bu değeri döndürmektir.

System.Boolean tipini ele almak gerekirse, bu tipte tanımlanan bir değişken içerisinde sadece True ve False değerleri saklanabilir. Benzer şekilde System.Int32 tipi ile sadece sayısal değerler saklanabilir.

Değer tiplere, değersizleşme özelliği katılması ile, ilgili değer tipine null değeri de atanabilmektedir. Özellikle veritabanı tablo işlemlerinde sık kullanılan, tamsayı bir alanın kullanıcı tarafından boş geçilmesi, böylece alan değerinin null olması benzeri bir durumdur.

System.Nullable sınıfı ile yönetilen değersizleşme işleminde yapılan, tanımlama gerçekte System.Nullable tanımıdır. CLR, çalışma anında yapılan tanımda kullanılan tipe göre otomatik olarak gerekli işlemleri yapmaktadır.

Aşağıda değersizleşebilen tipler ile ilgili çeşitli örnekler gösterilmiştir.

C#

**bool** b = **true**;

// bool b = null derleyici hatası verecektir.

**bool**? bn = **null**;

// bn içerisinde bir değer içeriyor ise

**if** (bn.HasValue)

Console.WriteLine("Değer içermektedir. Değer: {0}", bn);

**else**

Console.WriteLine("Değer içermemektedir");

bn = b;

**if** (bn.HasValue)

Console.WriteLine("Değer içermektedir. Değer: {0}", bn);

**else**

Console.WriteLine("Değer içermemektedir");

VB.NET

**Dim** b **As** **Boolean** = **True**

' Dim b As Boolean = nothing derleyici hatası verecektir.

**Dim** bn **As** Nullable(**Of** **Boolean**) = **Nothing**

' bn içerisinde bir değer içeriyor ise

**If** bn.HasValue **Then**

Console.WriteLine("Değer içermektedir. Değer: {0}", bn)

**Else**

Console.WriteLine("Değer içermemektedir")

**End** **If**

bn = b

**If** bn.HasValue **Then**

Console.WriteLine("Değer içermektedir. Değer: {0}", bn)

**Else**

Console.WriteLine("Değer içermemektedir")

**End** **If**

Kod 38, Değersizleşebilen Tip Örnekleri

C# için değer tipinin değersizleştirilebilmesi, tip tanımına eklenen ? karakteri, VB.NET için bu tanım Nullable kelimesi ile yapılmaktadır.

Değersizlik kontrolü C# için ?? operatörü ile yapılabilir. Örneğin  
  
**bool**? b3 = bn ?? **false**;  
  
Yukarıdaki işlem ile b3 değişkenine bn geçerli bir değer içeriyorsa içerdiği değer, içermiyorsa false değeri atanacaktır. Aynı işlemi VB.NET ve C# ile sırasıyla aşağıdaki şekilde de yapmak mümkündür.  
  
**Dim** b3 **As** Nullable(**Of Boolean**) = IIf(bn.HasValue, bn.Value, **False**)

**bool**? b3 = bn.HasValue ? bn.Value : **false**;

Sayılabilir Tipler

Sayılabilir tipler (ing. Enumarated Types) sonlu sayıda değeri gruplamak için kullanılabilecek yapılardır.

Aşağıda örnek sayılabilir tip tanım ve kullanımları gösterilmiştir.

C#

**public** **enum** TemelRenk {

Kırmızı,

Yeşil,

Mavi

}

// renk isminde, TemelRenk tipinde değişken tanımla ve ilk değerini ata.

TemelRenk renk = TemelRenk.Kırmızı;

// TemelRenk tipinde olan renk isimli değişkenin karakter dizi (string)  
// karşılığını s isimli değişkene ata.

**string** s = renk.ToString();

// Karakter dizi değerini işleyerek sayılabilir tip dönüşümü yapmak

// için kullanılmalıdır.

TemelRenk renk2 = (TemelRenk)Enum.Parse(**typeof**(TemelRenk), "Yeşil");

**int** x = (**int**)renk2;

VB.NET

**Public** **Enum** TemelRenk

Kırmızı

Yeşil

Mavi

**End** **Enum**

' renk isminde, TemelRenk tipinde değişken tanımla ve ilk değerini ata.

**Dim** renk **As** TemelRenk = TemelRenk.Kırmızı

' TemelRenk tipinde olan renk isimli değişkenin karakter dizi (string)  
' karşılığını s isimli değişkene ata.

**Dim** s **As** **String** = renk.ToString()

' Karakter dizi değerini işleyerek sayılabilir tip dönüşümü yapmak

' için kullanılmalıdır.

**Dim** renk2 **As** TemelRenk = [**Enum**].Parse(**GetType**(TemelRenk), "Yeşil")

**Dim** x **As** **Integer** = renk2

Kod 39, Sayılabilir Tip Örnekleri ve Kullanımları

Yukarıdaki örnekte, TemelRenk adında sayılabilir tip tanımı yapılmış, tip üyeleri Kırmızı, Yeşil ve Mavi olarak belirtilmiştir.

Sayılabilir tipler üye değerleri, eğer tanımlanmamış ise, tamsayı değeri olarak 0'dan başlarlar ve birer artarak tanım sırasına göre değer alırlar. Yukarıdaki örnekte x değişkeni 1 değerini alacaktır.

Sayılabilir tiplerin daha ileri seviye kullanımı, her bir sayılabilir tip üyesine sayısal değer atamak ve ikil işlemlerde (ing. Bitwise Operations) kullanmaktır.

Aşağıda bu işlemle ilgili örnek kod gösterilmiştir.

C#

[Flags]

**public** **enum** Hak {

Okuma = 1,

Yazma = 2,

Silme = 4,

Tümü = Okuma | Yazma | Silme

}

**class** Program {

**private** **static** **void** hakYazdır(Hak hak) {

**if** ((hak & Hak.Tümü) == Hak.Tümü)

Console.WriteLine("Tüm haklar mevcut");

**else**

{

**if** ((hak & Hak.Okuma) == Hak.Okuma)

Console.WriteLine("Okuma hakkı mevcut");

**if** ((hak & Hak.Yazma) == Hak.Yazma)

Console.WriteLine("Yazma hakkı mevcut");

**if** ((hak & Hak.Silme) == Hak.Silme)

Console.WriteLine("Silme hakkı mevcut");

}

}

**static** **void** Main(**string**[] args) {

Hak hak1 = 0;

Console.WriteLine("Hak1:");

hakYazdır(hak1);

Hak hak2 = Hak.Okuma | Hak.Silme;

Console.WriteLine("Hak2:");

hakYazdır(hak2);

Hak hak3 = Hak.Tümü;

Console.WriteLine("Hak3:");

hakYazdır(hak3);

Hak hak4 = Hak.Tümü & ~Hak.Okuma;

Console.WriteLine("Hak4:");

hakYazdır(hak4);

Hak hak5 = Hak.Okuma | Hak.Silme | Hak.Yazma;

Console.WriteLine("Hak5:");

hakYazdır(hak5);

Console.ReadLine();

}

VB.NET

<Flags()> \_

**Public** **Enum** Hak

Okuma = 1

Yazma = 2

Silme = 4

Tümü = Okuma **Or** Yazma **Or** Silme

**End** **Enum**

**Class** Program

**Private** **Shared** **Sub** hakYazdır(**ByVal** hak **As** Hak)

**If** ((hak **And** hak.Tümü) = hak.Tümü) **Then**

Console.WriteLine("Tüm haklar mevcut")

**Else**

**If** ((hak **And** hak.Okuma) = hak.Okuma) **Then**

Console.WriteLine("Okuma hakkı mevcut")

**End** **If**

**If** ((hak **And** hak.Yazma) = hak.Yazma) **Then**

Console.WriteLine("Yazma hakkı mevcut")

**End** **If**

**If** ((hak **And** hak.Silme) = hak.Silme) **Then**

Console.WriteLine("Silme hakkı mevcut")

**End** **If**

**End** **If**

**End** **Sub**

Kod 40, İleri Seviye Sayılabilir Tip Kullanımı

Örnek uygulama çıktısı aşağıdaki gibi olacaktır.

Hak1:

Hak2:

Okuma hakkı mevcut

Silme hakkı mevcut

Hak3:

Tüm haklar mevcut

Hak4:

Yazma hakkı mevcut

Silme hakkı mevcut

Hak5:

Tüm haklar mevcut

Listelenen kaynak kodlar \Bölüm4\Enums klasöründe bulunmaktadır.

Sayılabilir üye değerleri verilirken 1, 2, 4, 8 gibi 2'nin üsleri şeklinde vermek üye değerlerinin birbirleri ile karışmamasını sağlayacaktır. Sayılabilir üyeleri için 0 değeri özel anlam ifade edecek şekilde kullanılabilir. Yukarıdaki örnekte 0 değeri, örneğin, kullanıcının hiçbir hakkı olmadığını ifade edebilmektedir. Böylece hak olmadığını belirtir yeni bir üye eklemeye gerek kalmamaktadır.

Sayılabilir tiplerin yukarıdaki şekilde olduğu gibi en etkin biçimde kullanılması, ikil işlem operatörleri hakkında bilgi sahibi olmayı gerektirmektedir. Kitabın ilerleyen bölümlerinde bu konu hakkında detaylı bilgi sunulacaktır.   
  
Bir sayılabilir tip değişkeni örnekten de görülebileceği üzere içerisinde birden fazla sayılabilir üye değerini saklayabilmektedir. Sayılabilir tipin bu niteliği kazanabilmesi için Flags özniteliğinin (ing. Attribute) sayılabilir tip tanımında bulunması gerekmektedir. Öznitelikler ile ilgili geniş bilgi kitabın ilerleyen bölümlerinde verilecektir.  
  
Sayılabilir tipin Flags özniteliği ile donatılması, sayılabilir tipte olan değişkenlerin geçersiz değerleri de alabilmesini beraberinde getirmektedir. Yukarıdaki örnekte Hak2 değişkenine 1566 değeri atamak mümkün olmaktadır. Sayılabilir tiplerin Flags özniteliği ile donatılması durumunda, değerlerin kullanılmadan önce geçerlilik kontrolü yapılması önerilmektedir.

Tüm sayılabilir tipler System.Enum sınıfını ata sınıf olara kabul etmektedirler. Bu sınıf içerisinde tanımlı olan metot ve özellikler tanımlanan tüm sayılabilir tipler için geçerlidir.

Aşağıda System.Enum sınıfı temel üye bilgileri listelenmiştir.

| Üye Adı | Açıklama |
| --- | --- |
| GetNames | Sayılabilir tip üye adlarını almak için kullanılmalıdır. |
| Parse | Karakter dizisi değerinden sayılabilir tip dönüşümü yapmak için kullanılmalıdır. |
| IsDefined | Sayılabilir tip içerisinde belirtilen değerin tanımlanıp tanımlanmadığını döndürür. Özellikle geçerlilik kontrolü yaparken kullanılmalıdır. |
| Format | Sayılabilir tip değerinin karakter dizisi karşılığını döndürür. Örneğin **string** s = **Enum**.Format(**typeof**(Hak), hak2, "G");  // veya sayısal değer olarak almak için **string** s = **Enum**.Format(**typeof**(Hak), hak2, "D"); |

Tablo 15, System.Enum Sınıfı Temel Üye Bilgileri

Yapı Tipleri

Yapı tipleri ile hedeflenen, içerisinde birbirleri ile ilişkili bir veya birden çok değişkeni içeren gruplayıcı tipler oluşturmaktır.

Yapılar, nesne yönelimli programlama kapsamında sınıflara benzer özelliklere sahip olmasına rağmen sınıflardan en önemli farkları yapıların değer tipleri olmasıdır.

Aşağıda örnek tanım ve kullanımları gösterilmiştir.

| C# | VB.NET |
| --- | --- |
| // Nokta isminde yapı tipi tanımla.  **public** **struct** Nokta {  **public** **int** X;  **public** **int** Y;  **public** Nokta(**int** x, **int** y) {  **this**.X = x;  **this**.Y = y;  }  **public** **override** **string** ToString() {  **return** **string**.Format  ("{0},{1}", X, Y);  }  }  // Kullanımı  Nokta dot1, dot2;  dot1.X = 12;  dot1.Y = 5;  dot2 = **new** Nokta(4, 6);  Console.WriteLine(dot1.ToString()); | ' Nokta isminde yapı tipi tanımla.  **Public** **Structure** Nokta  **Public** X **As** **Integer**  **Public** Y **As** **Integer**  **Public** **Sub** **New**(**ByVal** x **As** **Integer**, \_  **ByVal** y **As** **Integer**)  **Me**.X = x  **Me**.Y = y  **End** **Sub**  **Public** **Overrides** **Function**  ToString() **As** **String**  **Return**  **String**.Format("{0},{1}", X, Y)  **End** **Function**  **End** **Structure**  ' Kullanımı  **Dim** dot1, dot2 **As** Nokta  dot1.X = 12  dot1.Y = 5  dot2 = **New** Nokta(4, 6)  Console.WriteLine(dot1.ToString()) |

Kod 41, Yapı Örnekleri

Yapılar konusunda detaylı açıklama kitabın bir sonraki bölümünde sınıflarla birlikte verilecektir.

Operatörler

Operatörler özel niteliği olan fonksiyonlardır. Operandlar, operatörler tarafından kullanılan parametre değerleridir.

Şekil 8, Operatör ve Operandlar

Operatörleri temel olarak ikili (ing. Binary) ve tekli (ing. Unary) olarak gruplamak mümkündür. İkili operatörler, iki adet operand ile anlam ifade eden operatörlerdir. Matematiksel toplama, çıkarma veya karakter dizi birleştirme operatörlere ikili operatörlere örnek olarak verilebilir.

Tekli operatörler, sadece bir operand ile işlem yapan operatörlerdir. Negatiflik ve koşullu Değil (ing. Conditional Not) operatörleri tekli operatörlere örnek olarak verilebilir.

Aşağıda CLR içerisinde kullanılabilecek önemli operatörler ve açıklamaları listelenmiştir.

| Operatör | Açıklama | C# | VB.NET |
| --- | --- | --- | --- |
| Eşitlik | İki operand değerinin birbirleri ile eşit olduklarının kontrolü için kullanılmalıdır. | == | = |
| Eşitsizlik | İki operand değerinin birbirleri ile eşit olmadıklarının kontrolü için kullanılmalıdır. | != | <> |
| Negatif | İlgili operand değerinin -1 ile matematiksel çarpım değerini hesaplamak için kullanılmalıdır. | - | - |
| Atama | Sol operand değerine sağ operand değerini atamak amacıyla kullanılması gereken operatördür. | = | = |
| Büyüklük | Operand tiplerine göre, sol operandın sağ operanddan daha büyük olup olmadığını kontrol etmek amacıyla kullanılması gereken operatördür. | > | > |
| Büyük Eşitlik | Operand tiplerine göre, sol operandın sağ operanddan daha büyük veya eşit olup olmadığını kontrol etmek amacıyla kullanılması gerken operatördür. | >= | >= |
| Küçüklük | Operand tiplerine göre, sol operandın sağ operanddan daha küçük olup olmadığını kontrol etmek amacıyla kullanılması gerken operatördür. | < | < |
| Küçük Eşitlik | Operand tiplerine göre, sol operandın sağ operanddan daha küçük veya eşit olup olmadığını kontrol etmek amacıyla kullanılması gerken operatördür. | <= | <= |
| Toplama | Operand tiplerine göre matematiksel toplama veya karakter dizileri için birleştirme operatörüdür. | + | +, & [[1]](#footnote-1) |
| Çıkarma | Matematiksel çıkarma operatörüdür. | - | - |
| Bölme | Matematiksel bölme operatörüdür. | / | / |
| Çarpma | Matematiksel çarpma operatörüdür. | \* | \* |
| Kalan Bölme | Tamsayı bölme işleminde kalan değeri hesaplamak için kullanılması gereken operatördür. | % | Mod |
| Üzeri | Sol operand değeri, sağ operand üzeri değeri ifade eder. Örneğin 2^3 gibi. | [[2]](#footnote-2) | ^ |
| Tamsayı bölme | Tamsayı bölem işlemlerinde kullanılmalıdır. | / | \ |
| Tip Eşitliği | Operandın belirtilen tipte olup olmadığının kontrolü için kullanılmalıdır. Örneğin  **if** (s **is** **string**) {} | is | Is |
| Üye Girişi | Nesne özellik, alan ve metot gibi üye girişlerinde kullanılmalıdır. | . | . |
| Tip Değeri | Operandın tip bilgisini almak için kullanılmalıdır. Örneğin  **Typeof**(x) | typeof | TypeOf |
| Değersizlik | Değersizleşebilen tipin değer içerip içermediğini kontrol etmek, içermiyorsa istenilen değeri döndürmek için kullanılmalıdır. | ?? | [[3]](#footnote-3) |
| Ve / Veya / Değil | Koşullu ve, veya ve değil operatörleridir. | &&, ||, ! | And, Or, Not |
| İkil ve, veya, Xor, değil | İkil işlemlerde kullanılması gereken operatörlerdir. | &, |, ^, ~ | And, Or, Xor, Not |
| İkil sağ ve sol kaydırma | İkil işlemlerde sağ ve sol kaydırma (ing. Shift Left, Shift Right) amacıyla kullanılması gereken operaratörlerdir. | <<, >> | <<, >> |
| Artır / Eksilt Eşitle | Operand değerini bir artırır / eksiltir ve operanda atar | ++, -- | [[4]](#footnote-4) |

Tablo 16, Operatörler ve Açıklamaları

Yukarıdaki listede yer almamakla birlikte, aşağıdaki kullanımlar gerek C#, gerek VB.NET için geçerlidir.

j += 12; // j = j + 12

i /= 12; // i = i / 12

Kod 42, Farklı operatör kullanımları

**Operatör Öncelikleri**

Anlatımların değer hesaplaması derleyicilerin operatör öncelik kuralları ile yönetilmektedir.

Genel kural olarak tekli operatörlerin ikili operatörler üzerinde, benzer şekilde bölme ve çarpma operatörlerinin toplama ve çıkarma üzerinde önceliği bulunmaktadır.

Operatör önceliği ile ilgili iki adet kod parçası aşağıda gösterilmiştir.

**Dim** i **As** **Integer** = 12

**Dim** j **As** **Integer** = 5

**Dim** k **As** **Integer** = i + j / 2 - **Not** i

Console.WriteLine(k)

**Dim** i **As** **Integer** = 12

**Dim** j **As** **Integer** = 5

**Dim** k **As** **Integer** = (i + j) / 2 - **Not** i

Console.WriteLine(k)

Kod 43, Operatör Önceliği

Kod parçası çıktısı aşağıdaki şekilde olacaktır.

28

22

Operatör önceliği, yanlış kullanılması durumunda bir çok yazılımda olası hataları beraberinde getiren konulardan birisidir. Genel kural olarak, programcıların öncelik konusunda emin olmadıkları durumda parantez kullanmaları önerilmektedir.

C# için ++ ve – – operatörleri sık kullanılan operatörlerdir. Bu operatörler sırasıyla operand değer artırımı ve değer eksiltme işlemlerini yapmaktadır.  
  
Operatörün operand öncesi kullanımı operand değerinin bir artırılması / eksiltilmesi ve artırılmış / eksiltilmiş değerin döndürülmesini, operatörün operand sonrası kullanımı operand değerinin benzer şekilde bir artırımı / eksiltmesi ancak değiştirilmemiş değerin döndürülmesini sağlamaktadır. Örneğin  
  
**int** x = 12;  
**if** (++x == 13) // Değer True dönecektir, x değeri 13 olacaktır.  
// veya  
**if** (x++ == 12) // Değer True dönecektir, x değeri 13 olacaktır.

**Anlatım Değerlerinin Hesaplanması**

Derleyicilerin kod optimizasyonu aşamasında, gerekli görülmeyen bazı anlatım / alt anlatım değerleri hesaplanmayabilir. Örneğin koşullu Veya (||, Or) operatörü için, operand değerlerinden herhangi birinin değerinin mantıksal doğru dönmesi, koşullu Ve (&&, And) operatörü için herhangi bir operand değerinin mantıksal yanlış dönmesi anlatım değeri sonucu için yeterlidir.

Aşağıda bununla ilgili bir örnek gösterilmiştir.

**int** x = 12, y = 15;

**if** (y == 2 && ++x != 5) {

//

}

Console.WriteLine(x);

**if** (y == 2 || ++x != 5) {

//

}

Console.WriteLine(x);

Kod 44, Anlatım Değerlerinin Hesaplanması

Yukarıdaki örnekte x değişken değeri iki kez artırılıyor olarak gözükmesine rağmen çıktı aşağıdaki gibi olacaktır.

12  
13

Derleyici, koşullu Ve işleminde y == 2 anlatım değerinin false olarak dönmesi ardından ++x != 5 anlatım değerini hesaplamamış, böylece x değeri artırılmamıştır.

Bu veya benzer durumlarda değer hesaplama işleminin hangi sırayla yapıldığı büyük önem kazanmaktadır. Genel kural olarak, hesaplama işlemi ikili operatörler için soldan sağa şeklinde yapılmaktadır. Buna rağmen bazı operatörler için anlatım değeri hesaplama işlemi sağdan sola şeklinde yapılabilir. Örneğin

**int** i = 12;

**int** j, k;

i = j = k = i + 1;

Console.WriteLine("i={0}, j={1}, k={2}", i, j, k);

Kod 45, Anlatım Değerlerinin Sağdan Başlayarak Hesaplanması

Yukarıdaki kod parçası çıktısı aşağıda gösterilmiştir.

i=13, j=13, k=13

Anlatım değerlerinin hesaplanması işlemi sırasında, anlatım içerisinde olan değişkenlerin değerlerinin değişmesi veya fonksiyon çağrımı sonucu farklı işlemler yapılıyor olması anlatımlara ek etkiler yaratabilmesi (ing. Side Effects) özelliğini katmaktadır. Bu durumda, yukarıdaki örneklerde olduğu gibi çalışma anında anlatım değerlerinin hesaplanmayabilir nitelikte olması, ek etkilerin de olmayacağı anlamına gelmektedir. Pratikte, yazılımların girdi bilgi içeriklerine göre bazen istendiği gibi, bazen hatalı çalışmasını beraberinde getirebilen bu durum konusunda programcıların dikkatli olmaları önerilmektedir.

**İkil İşlemler**

İkil işlemler, sayısal değerler için her bir ikil değer üzerinde işlem yapılabilmesini sağlamaktadır.

Örneğin 37 sayısı ikil değer karşılığı (Matematiksel ikili düzen) 00100101'dir. Benzer şekilde 148 ikil değer karşılığı 10010100'dir. İkil işlem operatörleri sayılarla bir bütün şeklinde değil, her bir ikil değer üzerinde ayrı ayrı hesaplama yaparak çalışmaktadırlar. Aşağıda ikil operatör açıklamaları ve örnekler listelenmiştir.

| Operatör | Açıklama | C# | VB.NET | Sonuç |
| --- | --- | --- | --- | --- |
| Ve | Her iki ikil değer 1 ise sonuç 1'dir, herhangi bir ikil değer 0 ise sonuç 0'dır. | 148 & 37 | 148 And 37 | 4 |
| Veya | Herhangi bir ikil değer 1 ise sonuç 1'dir, her ikisi sıfır ise sonuç sıfırdır. | 148 | 37 | 148 Or 37 | 181 |
| Değil | İkil değer 1 ise sonuç 0'dır, değer 0 ise sonuç 1'dir. | ~148 | Not 148 | -149 |
| Xor | İkil değerler birbirleri ile aynı ise 0, değilse 1 döndürür. | 148 ^ 37 | 148 XOr 37 | 177 |
| Sağ Kaydır | Belirtilen değer kadar 0 operanda soldan eklenir, sağdan eklenen miktarda düşürülür. (Kaydırma işlemi yapılır) | 148 >> 3 | 148 >> 3 | 18 |
| Sol Kaydır | Belirtilen değer kadar 0 operanda sağdan eklenir, soldan eklenen miktarda düşürülür. (Kaydırma işlemi yapılır) | 148 << 3 | 148 << 3 | 1184 |

Tablo 17, İkil Operatör Örnekleri

Yukarıda örneklenen operatörler çalışma genel prensibi ikil Ve ile XOr operatörü için aşağıdaki gösterilmiştir.

00100101 = 37 00100101 = 37  
10010100 = 148 10010100 = 148  
-------- (AND) -------- (XOR)  
00000100 = 4 10110001 = 177

Koşul Blokları

Mantıksal karşılaştırma blokları, programın belirtilen bir anlatım değerinin mantıksal doğru (True) veya mantıksal yanlış (False) olmasına göre farklı kod parçalarını çalıştırabilmesini sağlamaktadır.

Aşağıda bu amaçla kullanılabilecek farklı yapılar ile ilgili bilgiler aktarılmıştır.

**Eğer Bloğu**

Eğer bloğu, belirtilen bir anlatım değerine göre çalıştırılması istenen ifadeleri belirtmek için kullanılmalıdır.

Aşağıda çeşitli örnekler listelenmiştir.

| C# | VB.NET |
| --- | --- |
| **int** x = 12;  **if** (x == 0)  Console.WriteLine("...");  **if** (x == 12) {  Console.WriteLine("ilk ...");  Console.WriteLine("ikinci ...");  } **else** **if** (x > 5) {  Console.WriteLine("Üçüncü ...");  } | **Dim** x **As** **Integer** = 12  **If** x = 0 **Then**  Console.WriteLine("...")  **End** **If**  **If** x = 12 **Then**  Console.WriteLine("ilk ...")  Console.WriteLine("ikinci ...")  **ElseIf** x > 5 **Then**  Console.WriteLine("Üçüncü ...")  **End** **If** |

Kod 46, Eğer Bloğu Örnekleri

**Seçim Bloğu**

Seçim Bloğu, eğer bloğu benzeri, farklı diziliş yapısı ile anlatım değerine göre çalıştırılması istenen ifadeleri belirtmek için kullanılmaktadır.

| C# | VB.NET |
| --- | --- |
| **int** x = 12;  **switch** (x) {  **case** 12:  Console.WriteLine("12");  **break**;  **case** 15:  Console.WriteLine("15");  **break**;  // Varsayılan ifadeler  **default**:  Console.WriteLine("Diğer");  **break**;  } | **Dim** x **As** **Integer** = 12  **Select** **Case** x  **Case** 13  Console.WriteLine("13")  **Case** **Is** > 15  Console.WriteLine("> 15")  **Case** 16 **To** 25  Console.WriteLine("16 - 25")  ' Varsayılan ifadeler  **Case** **Else**  Console.WriteLine("Diğer")  **End** **Select** |

Tablo 18, Seçim Bloğu Örnekleri

**Seçenekler**

Seçenekler, belirtilen anlatım değeri sonucuna göre değer döndüren yapılardır.

Seçenekler, değeri hesaplanacak bir anlatım ve anlatım değerinin True olmasına göre belirtilen ilk anlatım, anlatım değerinin False olmasına göre ikinci anlatım değerini döndüren bir yapıdadır.

Seçenekler kaynak kodun okunabilirliğini artıran öğelerdir. Aşağıda çeşitli kullanımları örneklenmiştir.

| C# | VB.NET |
| --- | --- |
| **int** x = 12, y = 23;  // x, 12 değerine eşit ise y  // değerini, x 12 değerine eşit // değilse x++ anlatım sonucunu // döndür.  **int** j = x == 12 ? y : x++;  Console.WriteLine(j);  **int** k = j == 13 ? x : y;  Console.WriteLine(k);  Console.WriteLine(j);  Console.WriteLine(x); | **Dim** x **As** **Integer** = 12  **Dim** y **As** **Integer** = 23  ' x, 12 değerine eşit ise y  ' değerini, x, 12 değerine eşit ' değilse x değerini ' döndür.  **Dim** j **As** **Integer** = IIf(x = 12, y, x)  Console.WriteLine(j)  **Dim** k **As** **Integer** = IIf(j = 13, x, y)  Console.WriteLine(k)  Console.WriteLine(j)  Console.WriteLine(x) |

Kod 47, Seçenek Örnekleri

Kod parçası çıktısı aşağıdaki gibi olacaktır.

23

23

23

12

Yukarıdaki C# kod parçasında ek etkisi olan x++ anlatımının göz ardı edildiğine, böylece x değerinin artırılmadığına dikkat edilmelidir.

Döngü Blokları

Döngüler, belirli bir bloğun, belirtilen kriterlere göre istenilen sayıda çalıştırılabilmesini sağlayan yapılardır.

Dört temel döngü

* For Döngüsü
* For Each Döngüsü
* While Döngüsü
* Do While Döngüsü

Olarak özetlenebilir.

**For Döngüsü**

Döngü yapısı kurulurken yineleme sayısının belirli olduğu durumlarda kullanılmalıdır. Aşağıda örnek kod parçaları listelenmiştir.

| C# | VB.NET |
| --- | --- |
| **for** (**int** i = 0; i < 5; i++)  Console.WriteLine(i);  **for** (**int** j = 0; j < 5; j += 2)  Console.WriteLine(j); | **For** i **As** Integer = 0 **To** 4  Console.WriteLine(i)  **Next**  **For** j **As Integer** = 0 **To** 4 **Step** 2  Console.WriteLine(j)  **Next** |

Kod 48, For Döngüsü Örnekleri

Yukarıdaki kod parçası sonucu aşağıdaki çıktı üretilecektir.

0

1

2

3

4

0

2

4

C# ile VB.NET arasındaki döngü sonu sabitine (5 ile 4) dikkat edilmelidir.

**İleri Seviye For Döngüsü Kullanımı**

For döngüsü, C# dili için, yukarıda listelenen tüm döngü tipleri ile yapılabilen işlemleri kapsayacak şekilde kullanılabilmektedir.

For döngüsü üç temel parametre bloğundan oluşmaktadır.

**for** ([A]; [B]; [C]) {

İfadeler;

}

| A | B | C |
| --- | --- | --- |
| Tanımlama bloğudur. Bu bölümde bir veya birden fazla değişken tanımlamak mümkündür.  Blok her zaman döngü başlamadan önce 1 kez çalıştırılır. | Döngü sonlanma kriterlerinin bulunduğu bloktur.  Blokta temel olarak döngüyü değeri false dönmesi durumunda sonlandıracak anlatım bulunur.  Döngü başlangıcında ve her bir yinelemeden sonra çalıştırılır. | Her bir yineleme sonucu çalıştırılan bloktur.  Bu blok döngü başlangıcında çalıştırılmaz, yinelemenin tamamlanması ardından çalıştırılır. |

Tablo 19, C# For Döngüsü Detayları

Döngü başlangıcında A bloğu çalıştırılır. Bu blokta değişken tanımlamaları veya fonksiyon çağrımları yapılır ve bu blok bir daha çalıştırılmaz.

B bloğu çalıştırılır ve dönen anlatım değerinin True olması sonucu döngüye başlanır.

Yineleme tamamlandıktan sonra C bloğu çalıştırılır, hemen ardından B bloğu çalıştırılır. Döngü, B bloğu anlatım değerinin False dönmesi sonucu tamamlanır, True dönmesi sonucu yineleme tekrarlanır.

C# dili için diğer döngü tiplerini esnek yapısı ile bir anlamda gereksiz kılan for döngüsü ile ilgili aşağıda çeşitli kod örnekleri gösterilmiştir.

**static** **bool** kontrolEt(**int** x) {

**return** x < 5 ? **true** : **false**;  
 // veya kısaca return x < 5;

}

**static** **void** yazdır(**ref** **int** x) {

Console.WriteLine(x++);

}

**static** **void** Main(**string**[] args) {

**for** (**int** i = 0, j = 5, m = 5; // <- A Bloğu

kontrolEt(i); // <- B Bloğu

i++, j += 2, yazdır(**ref** m)) // <- C Bloğu {

Console.WriteLine("i={0}", i);

}

}

Kod 49, C# For Döngüsü Örnekleri

Yukarıdaki kod parçası çıktısı aşağıda gösterilmiştir.

i=0

5

i=1

6

i=2

7

i=3

8

i=4

9

A, B ve C blokları zorunlu değildir. Sonsuz yinelenen döngü, örneğin, aşağıdaki gibi kurulabilir.  
for (;;) {}

Listelenen kaynak kodlar \Bölüm 4\Loops klasöründe bulunmaktadır.

**For Each Döngüsü**

For Each döngüsü, diziler üzerinde, for döngüsü benzeri başlangıç ve bitiş değerleri vermeksizin yineleme yapmak amacıyla kullanılmalıdır.

For Each döngüsü aynı zamanda IEnumarator ve IEnumarable arabirimi ile farklı nesneler üzerinde de işlem yapabilmektedir. Bu konu ile ilgili detaylı bilgi kitabın ilerleyen bölümlerinde aktarılacaktır.

Aşağıda döngü kullanımı ile ilgili örnekler listelenmiştir.

| C# | VB.NET |
| --- | --- |
| **string**[] diller = **new** **string**[]   { "C#", "VB.NET", "Delphi.NET" };  **foreach** (**string** dil **in** diller) {  Console.WriteLine(dil);  }  // Aynı işlem for döngüsü ile  **for** (**int** i=0; i<diller.Length; i++) {  Console.WriteLine(diller[i]);  } | **Dim** diller **As** **String**() = **New** **String**()  {"C#", "VB.NET", "Delphi.NET"}  **For** **Each** dil **As** **String** **In** diller  Console.WriteLine(dil)  **Next**  ' Aynı işlem for döngüsü ile  **Dim** i **As** **Integer**  **For** i = 0 **To** diller.Length - 1  Console.WriteLine(diller(i))  **Next** |

Kod 50, For Each Döngüsü Örnekleri

**While Döngüsü**

Yineleme işleminin belirli bir koşulun tamamlanmasına kadar tekrarlanması istenilen durumlarda While döngüsü kullanılmalıdır.

Aşağıda döngü kullanımı ile ilgili örnekler listelenmiştir.

| C# | VB.NET |
| --- | --- |
| **string**[] diller = **new** **string**[]   { "C#", "VB.NET", "Delphi.NET" };  **int** i = -1;  **while** (++i < diller.Length) {  Console.WriteLine(diller[i]);  } | **Dim** diller **As** **String**() = **New** **String**()   {"C#", "VB.NET", "Delphi.NET"}  **Dim** i **As** **Integer** = 0  **While** i < diller.Length  Console.WriteLine(diller(i))  i = i + 1  **End** **While** |

Tablo 20, While Döngüsü Örnekleri

While döngüsü, döngüye başlarken belirtilen anlatım değerini kontrol etme özelliği ile yineleme yapılıp yapılmayacağının belirsiz olduğu durumlarda kullanılmalıdır. Örneğin disk üzerindeki bir dosyanın satır bazlı okunması işleminde, benzer şekilde veritabanından kayıtların alınması gibi yineleme sayısı belirsiz, aynı zamanda veri olmaması durumunda yinelemenin hiç yapılmayacağı işlemlerde bu döngü tipi kullanımı tercih edilmelidir.

Aşağıdaki örnek, kullanıcıdan metin bazlı bir dosya ismi almakta ve dosya içeriğini ekranda göstermektedir.

Console.WriteLine("Dosya adı giriniz:")

**Dim** fileName **As** **String** = Console.ReadLine()

**If** **Not** **String**.IsNullOrEmpty(fileName) **And** File.Exists(fileName) **Then**

**Dim** reader **As** StreamReader = **New** StreamReader(fileName, **True**)

**While** **Not** reader.EndOfStream

Console.WriteLine(reader.ReadLine())

**End** **While**

reader.Close()

**End** **If**

Kod 51, While Döngüsü Kullanımı

Dosya içeriğinin boş olması durumunda reader.EndOfStream anlatımı False döneceği için döngüye hiç girilmeyecektir. Benzer işlem C# için tek bir for döngüsü ile aşağıdaki şekilde yapılabilir.

**for** (StreamReader reader = **new** StreamReader(fileName, **true**);

!reader.EndOfStream;

Console.WriteLine(reader.ReadLine()));

Kod 52, For ve While Döngüsü

Listelenen kaynak kodlar \Bölüm 4\Loops klasöründe bulunmaktadır.

**Do While Döngüsü**

While döngüsü benzeri olan Do While döngüsü, yineleme işleminin en az bir kez yapılmasının istendiği durumlarda kullanılmalıdır.

Aşağıda döngü kullanımı ile ilgili örnekler listelenmiştir.

| C# | VB.NET |
| --- | --- |
| **string**[] diller = **new** **string**[];  diller= { "C#", "VB.NET",   "Delphi.NET" };  **int** i = 0;  **do** {  Console.WriteLine(diller[i]);  } **while** (++i < diller.Length); | **Dim** diller **As** **String**()  diller == **New** **String**()   {"C#", "VB.NET", "Delphi.NET"}  **Dim** i **As** **Integer** = 0  **Do**  Console.WriteLine(diller(i))  i = i + 1  **Loop** **While** (i < diller.Length)  i = 0  **Do**  Console.WriteLine(diller(i))  i = i + 1  **Loop** Until (i >= diller.Length)  i = 0  **Do** **Until** i >= diller.Length  Console.WriteLine(diller(i))  i = i + 1  **Loop** |

Kod 53, Do While Döngüsü Örnekleri

**Döngü Çıkışları**

Bazı durumlarda yineleme işlemi devam ederken döngü anlatım değeri kontrolü tamamlanmadan döngü içerisinde iken ilgili bloğun ve döngünün tamamlanması istenebilir. Benzer şekilde, döngü içerisinde iken bloğun içerdiği tüm ifadeler tamamlanmadan bir sonraki yinelemenin başlaması gerekebilir.

| İşlem | C# | VB.NET |
| --- | --- | --- |
| Döngüden Çıkış | break | Exit For  Exit While  Exit Do |
| Döngüye Devam | continue | continue |

Tablo 21, Döngü İşlemleri

Aşağıdaki örnekte en fazla 1000 adet rasgele sayı (ing. Random Number) üretilmektedir. Her 20'lik dilimde kullanıcıdan işlem durumu hakkında girdi istenmektedir.

Kullanıcının <d> seçeneğini kullanması durumunda döngü bloğu içerisinde bulunan Console.Clear() ifadesinin çalıştırılmadığına dikkat edilmelidir.

Random r = **new** Random();

**int** max = 1000, current = 0;

**while** (current++ < max) {

**int** number = r.Next();

Console.WriteLine(number);

**if** (current % 20 == 0) {

Console.WriteLine("[t: Ekran Temizle], [c: Çık], [d: Devam]:");

**string** s = Console.ReadLine().ToLowerInvariant();

**if** (s == "c")

**break**;

**else** **if** (s == "d")

**continue**;

Console.Clear();

}

}

Kod 54, Döngü İşlemleri Kod Örneği

With Bloğu

With bloğu, içerisinde bulunan ifadelerin belirli bir nesne ile çalıştırılabilmesini sağlamaktadır.

**With** Button1

.Text = ""

.Width = 12

**End** **With**

Kod 55, With Bloğu

Yukarıdaki kod birebir aşağıdaki ile aynı anlamı ifade etmektedir.

Button1.Text = ""

Button1.Width = 12

With sadece VB.NET tarafından desteklenmektedir.

Güvensiz Bloklar

Bilindiği gibi kaynak kodun derlenmesi ile IL kod yapısı oluşturulmaktadır. IL kod, CLS ve CTS ile standartları belirlenmiş yapıları ve tipleri içermektedir.

Yönetilen ortamda çalıştırılan IL kod daha üst seviyede işlemleri içermekte ve CLR tarafından yönetilmektedir. Yönetim kavramı, IL kod içeriğinin CLR tarafından kontrolü, güvenlik ve bütünlük prensiplerine uygunluk gibi bir dizi kuralı içermektedir.

Yönetilen ortamda çalışıyor olmanın bir dezavantajı alt seviye işlemlerin desteklenmemesidir. Örneğin, yönetilen ortamda hafıza yönetimi tamamen CLR tarafından yapılmakta, uygulamanın belirli bir hafıza alanını ayırması ve istediğinde boşaltması, işaretçi tip kullanımı gibi alt seviye işlemler desteklenmemektedir.

C#, programcının .NET platformu içerisinde daha alt seviye işlemler yapabilmesine kısıtlı destek vermektedir. Unsafe blokları içerisinde yapılabilen bu işlemler CLR tarafından güvensiz olarak nitelenmektedir.

Güvensizlik kavramı metotlar, tipler ve bloklar için geçerlidir.

**unsafe static void** IntCopy(**int**\* kaynak, **int**\* hedef, **int** boyut) {

**int**\* p = kaynak;

**for** (int i = 0; i < boyut; i++) {

\*(hedef + i) = \*(p + i);

}

}

Kod 56, Güvensiz Kod Örnekleri

Yukarıdaki örnekte IntCopy isimli metot unsafe rezerve kelimesi ile güvensiz olarak belirtilmiş, böylece bu metot içerisinde işaretçi ve işaretçi aritmetiği kullanılabilmiştir.

Güvensiz blok kullanımı VB.NET tarafından desteklenmemektedir.  
  
Güvensiz blokların derlenebilmesi için /:Unsafe derleyici parametresi kullanılmalı veya VS.NET proje özellikleri penceresinde Allow Unsafe Code seçeneği işaretli olmalıdır.

Tipler ve Tip Dönüşümü

Tip dönüşümü (ing. Type Casting), farklı tiplerin birbirleri arasındaki dönüşüm kurallarını ifade etmektedir.

Örneğin 12 sayısal değeri, System.Integer, System.Byte, System.SByte ve bunlar gibi sayısal tip değer aralıklarına düşen her tipte geçerli bir değerdir.

Bu bölümde, farklı tipte olan değişkenlerin atamalarında yapılan otomatik tip dönüşümü, tipler ile ilgili genel kurallar ve tip dönüşümü hakkında bilgiler verilecektir.

**Tip Bilgisi**

Çalışma anında değişkenlerin hangi tipte oldukları ve tipler hakkında detaylı bilgi almak için GetType metodu ve System.Type sınıfı kullanılabilir.

| C# | VB.NET |
| --- | --- |
| **int** i = 0;  Type t = i.GetType();  Type t2 = **typeof**(**string**[]);  Console.WriteLine(t.Name);  Console.WriteLine(t2.Name);  Console.WriteLine(t.Assembly.FullName); | **Dim** i **As** **Integer** = 0  **Dim** t **As** Type = i.GetType()  **Dim** t2 **As** Type = **GetType**(**String**())  Console.WriteLine(t.Name)  Console.WriteLine(t2.Name)  Console.WriteLine(t2.Assembly.FullName) |

Kod 57, Type Sınıfı Örnekleri

Yukarıdaki kod parçası çıktısı aşağıda gösterilmiştir.

Int32

String[]

mscorlib, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089

Type sınıfı temel üye bilgileri ve açıklamalar aşağıda gösterilmiştir.

| Üye Adı | Açıklama |
| --- | --- |
| Name | Tip adını döndürür. Örneğin Boolean. |
| FullName | Tipin bütünleşik adını döndürür. Örneğin System.Boolean. |
| AssemblyQualifiedName | Tipin tanımlı olduğu Assembly ile birlikte bütünleşik adını döndürür. Örneğin System.Boolean, mscorlib, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089. |
| IsArray | Tipin dizi tipi olup olmadığını belirtir. Benzer şekilde IsEnum, IsPrimitive, IsValueType vb. özellikleri tip bilgisi kontrolü amacıyla kullanılabilir. |
| GetType | Çalışma anında, belirtilen bir karakter dizesinden tip bilgisi nesnesi oluşturmak için kullanılmalıdır. Örneğin  Type ti = Type.GetType("System.Int32");  **bool** b = **false**; Type t = b.GetType(); |

Tablo 22, System.Enum Sınıfı Temel Üye Bilgileri

**Tip Dönüşümü**

Tip dönüşümünü ikiye ayırmak mümkündür. Dolaylı tip dönüşümü (ing. Implicit Type Casting), derleyici tarafından yönetilen ve tip dönüşümü için gerekli kodun otomatik olarak arka planda oluşturulduğu tip dönüşümüdür.

Direkt tip dönüşümü (ing. Explicit Type Casting), tip dönüşüm operatörleri ve yapıları ile programcı tarafından yapılan tip dönüşümüdür.

Tip dönüşümü ile ilgili aşağıda temel bazı örnekler gösterilmiştir.

| C# | VB.NET |
| --- | --- |
| **int** i = 12;  **byte** b = 12;  **byte** b2 = (**byte**)i;  **object** o = i **as** **object**;  **object** o2 = **new** **object**();  **object** o3 = (**object**)i;  **int** i2 = b;  Console.WriteLine(o.GetType().Name);  Console.WriteLine(o2.GetType().Name);  Console.WriteLine(o3.GetType().Name);  Console.WriteLine(b.GetType().Name); | **Dim** i **As** **Integer** = 12  **Dim** b **As** **Byte** = 12  **Dim** b2 **As** **Byte** = **CType**(i, **Byte**)  **Dim** o **As** **Object** = **TryCast**(i, **Object**)  **Dim** o2 **As** **Object** = **New** **Object**()  **Dim** o3 **As** **Object** = **CType**(i, **Object**)  **Dim** i2 **As** **Integer** = b  Console.WriteLine(o.**GetType**().Name)  Console.WriteLine(o2.**GetType**().Name)  Console.WriteLine(o3.**GetType**().Name)  Console.WriteLine(b.**GetType**().Name) |

Tablo 23, Tip Dönüşüm Örnekleri

Kod parçası çıktısı aşağıdaki gibi olacaktır.

Int32

Object

Int32

Byte

Yukarıdaki kod parçasında i2 = b ifadesi ile dolaylı tip dönüşümü örneklenmiştir. Byte tipinin Integer tipine dönüşümünün yapıldığı bu işlemin tersinin yapılması, yani b = i2 şeklinde bir ifade, derleyici tarafından hata üretilmesine sebep olacaktır. Bunun sebebi Integer tipinin Byte tipini kapsayabilecek değerleri alabilmesidir. Bu ve benzeri durumlarda dolaylı tip dönüşümü yapmak mümkündür.

Direkt tip dönüşüm işlemi çalışma anında yapılmaktadır. Dönüşümü yapılamayacak tipler için çalışma anında hata üretilecektir.

C# için gerek () gerekse as rezerve kelimesi ile tip dönüşümü yapılabilir. Parantez ile yapılan tip dönüşümü, tip dönüşüm işlemi yapılamaz ise hata üretecektir. As ile yapılan tip dönüşümünde, dönüşüm işleminin başarısız olması durumunda null döndürülecektir.

Benzer şekilde VB.NET için CType veya DirectCast tip dönüşümü amacıyla kullanılabilir. Her iki yapıda da dönüşümün başarısız olması durumunda hata üretilecektir. TryCast, dönüşüm işlemi başarısız ise nothing döndürecektir.

Tip dönüşümü kuralları ve özellikleri nesne tabanı programlama konusunun önemli alt konularındandır ve kitabın ilerleyen bölümlerinde detaylarıyla aktarılacaktır.

VB.NET için DirectCast, tip dönüşüm performansı CType'a göre daha optimizedir ancak tip dönüşümü yapılacak nesneler arasında miraslık aranmaktadır.

Koşullu Derleme ve Derleyici Direktifleri

Gerçek derleme işlemi başlamadan önce ön işleyici tarafından (ing. Pre-Processor) kaynak kod analizi yapılmakta ve kaynak kod derleyiciye hazır hale getirilmektedir.

Kaynak kodun belirli kurallar ile işlenmesi ve derleyiciye hazırlanması adımı derleyici direktifleri ile yönetilmektedir. Koşullu derleme, derleyici direktifleri yardımıyla, ön işleyici tarafından kaynak kodun sadece bazı bölümlerinin derleme adımına aktarılabilmesini sağlamakta, böylece koşullu derleme yapılabilmektedir.

Koşullu derleme ile kod içerisinde bulunan koşul blokları birbirine karıştırılmamalıdır. Koşullu derleme, belirtilen koşulu sağlamayan kod bölümlerinin derlenmemesini sağlamakta, böylece IL kod içerisinde bu kod blokları bulunmamaktadır.

Derleyici direktifleri # (Diyez) karakteri ile başlamaktadır.

DEBUG, özel anlam ifade eden derleyici direktifidir. Bu direktif ile derleyici, hata ayıklama işlemi yapabilmek için gerekli yapıları hazırlamaktadır.

Derleyici direktifleri aynı zamanda, derleme işlemine herhangi bir etkisi olmayan ancak kaynak kod okunabilirliğini ve yönetilebilirliğini artıran, kod içerisindeki çeşitli mantıksal bölümleri görsel olarak ayırmak için de kullanılabilmektedir.

Aşağıda çeşitli derleyici direktifleri hakkında bilgiler sunulmuştur.

| Direktif | C# | VB.NET |
| --- | --- | --- |
| Koşul | #if  #else  #elif  #endif | #if  #else  #else if  #end if |
| Tanım | #define | #const |
| Mantıksal Gruplama | #region  #endregion | #region  #end region |

Tablo 24, Tip Dönüşüm Örnekleri

Derleyici direktifleri kullanımı konusunda aşağıda kod örnekleri gösterilmiştir.

| C# | VB.NET |
| --- | --- |
| #define DETAYLICIKTI  **namespace** DotNetKitabı {  …  #region "Tanımlar"  **int** i = 0;  **byte** j = 1;  #endregion  #if (DETAYLICIKTI && DEBUG)  Console.WriteLine(  "Tanımlamalar yapıldı");  #endif | #**Const** DETAYLICIKTI = **True**  #**Region** "Giriş Metodu"  **Shared** **Sub** Main(**ByVal** args()   **As** **String**)  #**If** DETAYLICIKTI **Then**  Console.WriteLine(".....")  #**End** **If**  **End** **Sub**  #**End** Region |

Kod 58, Derleyici Direktifleri Kullanım Örnekleri

Derleyici direktif değerleri komut satırı derlemelerinde /Define: parametresi ile veya VS.NET proje özellikleri penceresi ile düzenlenebilir.   
  
Örneğin  
vbc.exe KaynakDosya.vb /define:DEBUG=True,DETAYLICIKTI=True  
csc.exe KaynakDosya.cs /define:DETAYLICIKTI

Özet ve Öneriler

Oldukça teorik konulardan oluşan bu bölümde okuyucuya

* Ortak tip sistemi ve .NET temel kavramları hakkında bilgiler.
* Programlama kavramları.
* C# ve VB.NET temel dil yapısı ve özellikleri.
* Hakkında bilgiler verilmiştir. Aynı zamanda CTS içerisinde yer alan tiplerden bazılarının kullanımları ile ilgili somut bilgilerin de bulunduğu bölümün tamamlanması ile okuyucunun temel program yapısı, koşullar, döngüler, değişken tanımlama gibi temel programlama bilgisini kazandığı düşünülmektedir.
* Kitabın bir sonraki bölümünde daha ileri seviye konular ve özellikle nesne programlama teknikleri hakkında bilgiler verilecektir.

Bir sonraki bölüme geçmeden önce kısa bir tekrar yapılması önerilmektedir.

Toplumu gerçek amacına, gerçek mutluluğuna ulaştırmak için iki orduya gerek vardır. Biri vatanın hayatını kurtaran asker ordusu, diğeri ulusun geleceğini yoğuran bilim ordusudur. Bu ordulardan her ikisi de aynı derece gerekli, kıymetlidir, her ikisi de hayatidir. Ancak bilim ordusunun kıymet ve kutsallığını anlatmak için şunu söyleyeyim ki, bilim ordusu, ölen ve öldüren birinci orduya, niçin ölüp, niçin öldürdüğünü öğreten ordudur.

Mustafa Kemal Atatürk

Bölüm

5

Nesne Yönelimli Programlama

N

esne yönelimli programlama - NYP (ing. Object Oriented Programming) ve nesne yönelimli yaklaşım (ing. Object Oriented Approach) günümüzde uygulama yazılımları geliştirmek, tasarlamak ve analiz etmek için detaylarıyla bilinmesi gereken en önemli kavramlardan ve konulardan birisidir.

Klasik fonksiyonel programlama tekniklerine yeni bir bakış açısı getiren ve farklı nitelikler kazandıran NYP, sağlam ve esnek özellikleri ile bir yazılım projesinin analiz ve tasarım gibi farklı aşamalarında da uygulanabilir bir disiplin halini almıştır.

.NET programlama konusunda başarılı ve uzman olmak isteyen her programcının öncelikli adımının, teorik düzeyde ve her yönüyle NYP mantık ve kurallarını öğrenmesi, bu kuralları pratikte uygulayabilir olması ve geliştirebilmesi olarak özetlenebilir.

Bu bölümde, teorik düzeyde bilginin zorunlu olduğu ve somut proje deneyimleri ile bilgi düzeyinin daha da artırılabileceği NYP, tamamı nesne tabanlı olan .NET platformu ve C#, VB.NET dil yapısı ile birlikte tüm yönleri ile ele alınacaktır.

Başlangıç Bilgileri ve Öneriler

Genişletilmiş kavramsal ifadesi ile nesne yönelimli yaklaşım, yazılım projelerinin temel özet amacı olan "Mevcut Problemin Bilgisayar Destekli Çözümü veya İyileştirilmesi" konusuna problemin teşhisinden çözümün ortaya konmasına kadar olan tüm aşamalarda farklı bir disiplin ve bakış açısı ile yaklaşmayı gerektirmektedir.

Fonksiyonel programlama teknikleri değerlendirildiğinde, var olan somut problem ve bu problemin yazılım ile çözümlenmesi arasında gerek tanımsal gerekse yapısal farklar bulunmaktadır. Aşağıdaki örnek problem değerlendirildiğinde;

ABC derneği, klasik seçim sandığı ziyareti ile yapılan seçimlerin elektronik ortamda yapılmasını istemektedir. Seçmenler öncelikle yöneticiler tarafından tanımlanmalı, gerekli kimlik doğrulama ve yetkilendirme bilgileri seçmenlere atanmalı ve seçmenlerin internet üzerinden güvenli şekilde oy kullanmaları sağlanmalıdır. Oy sayım mekanizması ile tüm oylar sayılabilmeli ve sağlanacak denetim mekanizması ile sayılmış oyların doğruluğu ve geçerliliği denetlenebilmelidir.

Oldukça özet şekliyle ifade edilen problemde, ifade edilen kapsam değerlendirildiğinde aşağıda listelenmiş olan işlevler belirlenebilir.

* Yöneticinin seçmen tanımlaması
* Seçmenlere kimlik doğrulama bilgileri atanması
* Seçmenlerin oy kullanması
* Seçim sonunda oyların sayılması
* Sayılan oyların denetlenmesi

İşlevler bazında yaklaşıldığında ve işlevlerin yazılıma yansıtılması klasik programlama teknikleri ile değerlendirildiğinde, aşağıda listelenen bazı fonksiyon ve yapılar kullanılıyor olacaktır.

TSecmenBilgisi = record

SecmenNo: Integer;

...

End;

procedure SecmenTanimla(SecmenBilgisi: TSecmenBilgisi);

void SecmenAta(int secmenNo);

sub OyKullan(SecmenNo as Integer, Oy as String);

...

Kod 59, Klasik Programlama Yaklaşım Örnekleri

Yukarıdaki örnekte dikkat edilmesi gereken konulardan birisi, yazılımın kullandığı tiplerin tanımlanması ve çeşitli fonksiyonlara parametreler iletilerek problemlerin çözülmesidir. Çözüm yöntemi değerlendirildiğinde, gerçek dünya varlıkları ile yazılımda bulunan tip tanım ve fonksiyonlarının tamamen örtüşmediği, yazılım çözümünün kendi içerisinde bir bütün olmasına rağmen gerçek problemin olduğu şekli ile yazılıma yansımadığı görülecektir.

Aynı problem nesne yönelimli yaklaşım disiplini ile değerlendirildiğinde, işlevlerin edilgin tanımlanması yerine gerçek dünya varlıkları ve bu varlıkların neler yapabileceği temel alınmalıdır.

Yazılım projelerinde bir standart olan ve üreticiler tarafından da benimsenen UML – Unified Modeling Language ile, gereksinimlerin belirlenmesinden yazılımın dağıtımına kadar olan bütün sürecin modellemesi yapılabilir. Aşağıda örnek kullanım senaryo modeli (Use Case Diagram) gösterilmiştir.

Şekil 9, Örnek UML Diyagramı

Belirtilen model irdelendiğinde, daha önce belirlenmiş işlevlerin daha somut olarak aktörler, yani sistem kullanıcıları, ve aktörlerin yapabilecekleri işlevler şeklinde özetlendiği görülecektir. Aynı analiz temel alınarak, örneğin, aşağıdaki kod yapısı oluşturulabilir.

public abstract class Kullanıcı {

...

}

public class Secmen: Kullanıcı {

void OyKullan();

}

public class SeçimYöneticisi: Kullanıcı {

void SecmenTanimla(Secmen secmen);

void ...

}

Kod 60, Örnek NYP Kaynak Kodu

Klasik çözüm ile karşılaştırıldığında, yukarıdaki kaynak kod ve kullanılan yapıların gerçek problem ile ne düzeyde örtüştüğü görülecektir.

Nesne yönelimli yaklaşım, problem analiz (Gereksinim Belirleme) ve tasarım süreçlerinde, işlev ve yapıların belirlenmesi ve belirlenen yapıların kaynak koda olabildiğince yansımasını sağlamaktadır. Böylece, geliştirilen yazılımın test süreci de kullanım senaryoları bazında yapılabilmektedir.

Kitabın konusu dışında olan ve birçok üretici tarafından kabul gören UML, yazılım projelerinin her adımında olabildiğince kullanılmalıdır.   
  
Nesne yönelimli yaklaşım değerlendirildiğinde, özellikle analiz, tasarım ve geliştirme aşamalarında teorik bilgi düzeyi yanında geçmiş deneyimin de iş kalitesine büyük etkisi bulunduğu unutulmamalıdır.

Programcıların şu ana kadar genel hatları ile aktarılmaya çalışılan yaklaşımı ve nesne yönelimli analiz ve düşünme tarzını benimsemeleri önerilmektedir.

Kavramlar

NYP üç temel gereksinim ve kavram üzerine kuruludur. Bunlar

* Bilgi Gizleme ve Koruma (ing. Information Hiding & Encapsulation).
* Miraslık (ing. Inheritance).
* Polimorfizm (ing. Polymorphism).

Herhangi bir platform veya programlama dilinin nesne tabanlı olduğundan söz edebilmek için asgaride yukarıdaki kavramları desteklemesi gerekmektedir.

Bilgi gizleme ve koruma, bir nesnenin içsel bütünlüğünü koruyabilmesini ve sadece istediği veriyi dışarıya açabilmesini ifade etmektedir.

Miraslık, bir nesnenin sağladığı işlevselliği kullanarak yeni nesneler üretebilme becerisini ifade etmektedir. Miraslık, özellikle tekrar kullanılabilirlik (ing. Re-Usability) konusunda önemli avantajlar sağlamaktadır.

Polimorfizm, farklı tiplere ait olan nesnelerin metot ve özellik çağrılarına tip bazında yanıt verebilme yeteneğini ifade etmektedir.

Sınıf ve Nesne

Sınıf ve nesne tanımları birbirlerine karıştırılabilen kavramlardır.

Sınıf, birbirleri ile ilişkili alan, özellik ve metotların tanımlandığı ve gerçekleştirimin de bulunabileceği tiptir.  
  
Nesne, bir sınıfın oluşturulmuş şeklidir. (Sınıf örneğidir, ing. Instance of a class)

Daha somut olmak gerekirse, yukarıdaki örnekte Secmen adında bir sınıf tanımı yapılmıştır. Yapılan bu tanım CTS içerisinde yapılabilecek dizi, sayılabilir tip benzeri bir tanımlamadır.

Aşağıdaki örnekte Secmen sınıfına ait bir nesne tanımı yapılmakta ve nesne oluşturulmaktadır.

Secmen secmen = **new** Secmen();

Klasik programlama ile karşılaştırıldığında sınıf, tip tanımı, nesne, değişken tanımı olarak değerlendirilebilir.

Temelde farklı kavramları ifade etmelerine rağmen sınıf ve nesne neredeyse birbirlerine karışmış olarak kullanılmaktadır. Örneğin .NET içerisindeki temel sınıf tanımı System.Object olarak yapılmıştır. System.Object gerçekte bir sınıf olmasına rağmen adı nesnedir (Object).  
  
Her iki kelimenin ne anlam ifade ettiği bilinerek farklı kullanımlar doğal karşılanmalıdır.

Sınıf Tanımlama

Sınıf tanımı class rezerve kelimesi ile yapılmaktadır. Aşağıda örnek sınıf tanımları gösterilmiştir.

C#

**public** **class** WebSayfasi {

**private** **string** title;

**public** WebSayfasi(**string** title) {

**this**.title = title;

}

**public** **string** Title {

**get** {

**return** title;

}

}

}

VB.NET

**Public** **Class** WebSayfasi

**Private** \_title **As** **String**

**Public** **Sub** **New**(**ByVal** title **As** **String**)

**Me**.\_title = title

**End** **Sub**

**Public** **ReadOnly** **Property** Title() **As** **String**

**Get**

**Return** \_title

**End** **Get**

**End** **Property**

**end** **class**

Kod 61, Örnek Sınıf Tanımları

Bir sınıf tanımı aşağıdaki yapıları içerisinde bulundurabilir.

* Alan tanımları
* Özellik tanımları
* Metot tanımları

**İç İçe Sınıf Tanımı**

Bir sınıf içerisinde farklı bir sınıf tanımlaması yapılabilir.

| C# | VB.NET |
| --- | --- |
| **public** **class** WebSayfasi {  **class** Frame {  . . .  }  **private** **string** title;  . . .  } | **Public** **Class** WebSayfasi  **Class** Frame  . . .  **End Class**  **Private** \_title **As** **String**  **. . .**  **end** **class** |

Kod 62, İç içe Sınıf Tanımları

Nesne Oluşturma

Nesne oluşturma new rezerve kelimesi ile yapılmaktadır. Aşağıda kod örneği gösterilmiştir.

| C# | VB.NET |
| --- | --- |
| // w1 isminde, WebSayfasi tipinde nesne // tanımı.  WebSayfasi w1;  // Tanımlanan w1 isimli nesnenin  // oluşturulması.  w1 = **new** WebSayfasi("B");  WebSayfasi w2; w2 = **new** WebSayfasi("B2");  Console.WriteLine(w1.Title); | ' w1 isminde, WebSayfasi tipinde nesne ' tanımı.  **Dim** w1 **As** WebSayfası  // Tanımlanan w1 isimli nesnenin  // oluşturulması.  w1 = **New** WebSayfasi("B")  **Dim** w2 **As** WebSayfasi w2 = **New** WebSayfasi("B")  Console.WriteLine(w1.Title) |

Kod 63, Nesne Oluşturma

Soyutlama

Soyutlama, tekrar kullanılabilirlik gibi bazı avantajları elde edebilmek için sınıfların olabildiğince gruplanmasını ve yeni sınıflar türetilmesini anlatan kavramdır. Yukarıdaki örnekte Seçmen sınıfı ata sınıf olarak Kullanıcı sınıfını kabul etmektedir, yani Seçmen, Kullanıcı sınıfını miras almaktadır. Burada yapılan bir derece soyutlamadır.

Soyutlama istenilen derecede artırılabilir. Örneğin aşağıda üç derece soyutlama yapılmıştır.

Şekil 10, Soyutlama

Soyutlama, nesne yönelimli yaklaşım disiplininde oldukça önemli bir konudur. Soyutlama derecesi, aynı zamanda ne düzeyde esneklik kazanıldığını da belirten bir değer olarak düşünülebilir. Soyutlama yapılırken aşağıdaki konular ve öneriler göz önüne alınmalıdır.

* Sadece gerektiği düzeyde soyutlama yapılmalıdır. Soyutlamanın çok ileri gitmesi gerek kodun okunabilirliğini gerekse performansı düşürecektir.
* Soyutlama yapılırken gerçek dünya nesneleri ve nesnelerin ortak özellikleri baz alınmalıdır. Başlangıç noktası olarak nesnelerin ortak özelliklerinin belirlenmesi ve ortak özelliklere göre soyutlama yapılması benimsenebilir.
* Ortak özelliklerin yanı sıra nesnelerin işlev ortaklıkları da soyutlama yaparken göz önünde bulundurulmalıdır.

Miraslık konusunun temelini oluşturulan soyutlama hakkında kitabın ilerleyen bölümlerinde farklı örneklemeler gösterilecektir.

Arabirimler

Arabirimler (ing. Interfaces), içerisinde gerçekleştirim (ing. Implementation) bulunmayan ve sadece işlevselliği tanımlayan yapılardır.

.NET içerisinde sık kullanılan ve oldukça ileri seviye NYP tekniklerini uygulayabilmeyi sağlayan arabirimlerin sağladığı avantajlardan birisi nesnelerin birbirleri ile haberleşirken karşılıklı sınıf detaylarını bilmeden haberleşmesini sağlamaktır. Özellikle farklı üreticiler tarafından üretilen yazılımların birbirleri ile haberleşmesinde, iki üreticinin belirli bir arabirimde mutabık kalması ile tüm gerçekleştirim detaylarının göz ardı edilebilmesi ile bu özellik oldukça faydalıdır.

Arabirim bazlı programlama (ing. Interface Based Programming) gibi bir kavramı oluşturabilecek kadar yazılımlar içerisinde yoğun kullanılabilen arabirim kavramı, sunmuş olduğu esneklik ve somut örneklerle ilerleyen bölümlerde detaylarıyla aktarılacaktır.

Sınıf soyutlaması ve arabirim kullanımı birbirleri ile karıştırılabilen konulardır. Sınıf soyutlaması ile hedeflenen ata sınıf içerisinde olabildiğince ortak işlevlerin gerçekleştiriminin yapılması ve tekrar kullanılabilirliğin en üst düzeyde sağlanmasıdır. Arabirim kullanımının daha az miktarda kod geliştirmeye direkt etkisi bulunmamaktadır. Arabirim kullanımı ile temel amaç niteliklerin varlıklardan bağımsız olarak tanımlanabilmesidir.

Arabirim Tanımı

Arabirim tanımı interface rezerve kelimesi ile yapılmaktadır. Aşağıda örnek arabirim tanımı gösterilmiştir.

C#

**public** **interface** IKarsilastirilabilir {

**int** Karsilastir(**object** dest);

}

VB.NET

**Public** **Interface** IKarsilastirilabilir

**Function** Karsilastir(**ByVal** dest **As** **Object**) **As** **Integer**

**End** **Interface**

Kod 64, Arabirim Tanımı

Üyeler

Üyeler sınıflar içerisinde bulunan alanlar, özellikler ve metotlardır. Üyeler, NYP bilgi gizleyebilme kuralı çerçevesinde dışarıya açık veya kapalı olabilir, soyut veya sanal olarak tanımlanabilir.

Üyeler, CTS içerisinde üye tip tanımı ile ifade edilmektedir.

Üye tanımı genişletilerek sayılabilir tipler, arabirimler ve sınıflar için bir bütün olarak kullanılabilir.

Sınıf ve Nesne Özellikleri

NYP disiplininin üç temel gereksinimi olan bilgi gizleme, miras alabilme ve polimorfizm çerçevesinde, temel düzeyden ileri seviye kullanım örneklerine kadar bu konu başlığında bilgiler verilecektir.

Temel Tanımlar

Bir sınıf içerisinde temel olarak

* Alan tanımlamaları.
* Özellik tanımlamaları.
* Metot tanımlamaları.

Yapılabilir. Bu yapılar Sınıf üyeleri olarak ifade edilmektedir. Nesnelere ait üyelere giriş . operatörü ile yapılmaktadır.

Alanlar

Alanlar, sınıf içerisinde tanımlı değişkenlerdir. Tanımlama kuralları değişken tanımlama kuralları ile benzerdir.

Aşağıda örnek alan tanımları gösterilmiştir.

| C# | VB.NET |
| --- | --- |
| **public** **class** Televizyon {  **private** **int** kanalSayisi;  **public** **string** marka = "TV Marka";  } | **Public** **Class** Televizyon  **Private** kanalSayisi **As** **Integer**  **Public** marka **As** **String** = "TV Marka"  **End** **Class** |

Kod 65, Sınıf Alan Tanımları

Alanların temel kullanım amacı nesnelerin bilgi saklamalarını sağlayabilmektedir.

Sınıf alanlarının olabildiğince dışarıya gizli olarak tanımlanması, dışarıdan erişimlerin özellikler yardımıyla yapılması önerilmektedir.

Yerel değişken tanımında olduğu gibi alanlar da tanımlama aşamasında ilk değer atamasını desteklemektedir.

Özellikler

Özellikler, alanlara benzer nitelikte, değerinin okunması ve değiştirilmesi durumunda kod gerçekleştiriminin yapılabileceği yapılardır.

Aşağıda örnek özellik tanımları gösterilmiştir.

C#

**public** **class** HesapMakinesi {

**protected** **string** marka = "";

**private** **string** renk = "Gri";

**public** **string** Marka {

**get** {

**if** **string**.IsNullOrEmpty(marka))

**return** "Markasiz";

**else** **return** marka;

}

}

**public** **string** Renk {

**get** { **return** renk; }

**set** { renk = value; }

}

}

VB.NET

**Public** **Class** HesapMakinesi

**Protected** \_marka **As** **String** = ""

**Private** \_renk **As** **String** = "Gri"

**Public** **ReadOnly** **Property** Marka() **As** **String**

**Get**

**If** **String**.IsNullOrEmpty(\_marka) **Then**

**Return** "Markasiz"

**Else**

**Return** \_marka

**End** **If**

**End** **Get**

**End** **Property**

**Public** **Property** Renk() **As** **String**

**Get**

**Return** \_renk

**End** **Get**

**Set**(**ByVal** value **As** **String**)

\_renk = value

**End** **Set**

**End** **Property**

**End** **Class**

Kod 66, Sınıf Özellik Tanımları

Özellik tanımı C# için özel bir kelime gerektirmemektedir. VB.NET için özellik tanımı Property rezerve kelimesi ile yapılmaktadır.

Özellikler aynı zamanda metot çağrımları gibi de değerlendirilebilir.

**Özellik Değerlerinin Alınması**

Özellik değerleri, tanım içerisinde bulunan Get bloğu içerisinde döndürülmelidir. Değerin döndürülmesi için Return rezerve kelimesi kullanılmalıdır.

Get bloğu içerisinde her türlü işlem yapılabilir. Yukarıdaki örnekte marka isimli alan değeri kontrol edilmekte, bu alanda değer bulunamazsa (string.IsNullOrEmpty) ön tanımlı değer döndürülmektedir.

Aşağıdaki örnek kod çalıştırıldığında

HesapMakinesi h = **new** HesapMakinesi();

Console.WriteLine(h.Marka);

Marka isimli özelliğin Get bloğu otomatik olarak çalıştırılacak ve return kelimesi ile döndürülen değer okunmuş olacaktır.

**Özelliklere Değer Atanması**

Alanlara ve değişkenlere değer atama ile aynıdır. Bir özelliğe değer ataması yapıldığında otomatik olarak set bloğu çalıştırılmaktadır. Atanmak istenen değer value yardımıyla alınabilir.

Aşağıdaki örnek kod çalıştırıldığında

HesapMakinesi h = **new** HesapMakinesi();

h.Renk = "Siyah";

Renk isimli özelliğin set bloğu otomatik olarak çalıştırılacak ve set bloğu içerisinde value yardımıyla "Siyah" karakter dizi değeri alınabilecektir.

**Salt Okunur Özellikler**

Sınıf içerisinde bir özelliğin sadece okunması ancak değer ataması yapılamaması istenebilir. Örneğin bir liste sınıfı değerlendirildiğinde, listeye eleman ekleme ve çıkarma ile değişen listenin toplam boyutunun okunabilmesi ancak dışarıdan değiştirilmesi doğru değildir. Bu durumda, örneğin Boyut özelliği salt okunur olarak tasarlanabilir.

C# için bir özelliğin salt okunur olmasını sağlamak için set bloğunu özellik içerisine eklememek yeterlidir. VB.NET için ReadOnly rezerve kelimesi ile özelliğin donatılması ve set bloğunun özellik tanımında bulunmaması gerekmektedir.

Yukarıdaki örnekte Marka isimli özellik salt okunurdur.

**Salt Yazılır Özellikler**

Özellik değerinin okunamaması ancak özelliğe değer atanabilmesini gerektiren durumlarda salt yazılır özellikler kullanılabilir.

C# için bir özelliğin salt yazılır olmasını sağlamak için get bloğunun bulunmaması yeterlidir. VB.NET için özellik WriteOnly ile donatılmalıdır.

**Varsayılan ve Parametreli Özellikler**

Sınıfın varsayılan özelliği, özellik adını kullanmadan ilgili özelliğe erişimi mümkün kılmaktadır. Sınıfa ait sadece bir adet varsayılan özellik bulunabilir ve varsayılan özellik parametreli (veya indeksli) özellik olmalıdır.

Çoğunlukla içerisinde farklı elemanlar içerebilen liste sınıflarında kullanılabilecek parametreli özellikler, özelliğin değerini okuma ve değer atama işlemlerinde, parametre değeri iletebilmeyi mümkün kılmaktadır.

Aşağıda varsayılan ve parametreli özellik örneği gösterilmiştir.

C#

**public** **class** Tablo {

**public** **object** **this**[**int** satir, **string** sutun] {

**get** {

. . .

}

**set** {

. . .

}

}

}

// Kullanım

Tablo t = **new** Tablo()*;*

Console.WriteLine(t[0, "Numara"])*;*

t[1, "Ad"] = 12*;*

VB.NET

**Public** **Class** Tablo

**Default** **Public** **Property** Item(**ByVal** satir **As** **Integer**, \_  
 **ByVal** sutun **As** **String**) **As** **Object**

**Get**

. . .

**End** **Get**

**Set**(**ByVal** value **As** **Object**)

. . .

**End** **Set**

**End** **Property**

**End** **Class**

'Kullanım

**Dim** t **As** Tablo = **New** Tablo()

Console.WriteLine(t(0, "Numara"))

t(1, "Ad") = 12

Kod 67, Varsayılan ve İndeksli Özellik Tanımları

C# için varsayılan özellik tanımı this rezerve kelimesi ile, VB.NET için default rezerve kelimesi ile yapılmaktadır. Yukarıdaki örnekte satir ve sutun isimli parametrelere iletilen değerler get ve set blokları içerisinde kullanılabilir.

Özellikler, değer atama ve okuma yönleri ile alanlara, get ve set blokları içerisinde kod çalıştırılabilmesi ile de metotlara benzemektedir.  
  
Genel kural olarak, bir özelliğin işlevi sadece herhangi bir alan değerine değer atama ve/veya değer okuma olsa dahi, alan tanımlarının public ile direkt dışarı açılmaması, olabildiğince alanları private veya protected tanımlayıp, gerekiyorsa public özellikler kullanılarak dışarıya açılması önerilmektedir. C#'ın sağladığı büyük küçük harf ayrımı yardımıyla alan tanımları küçük harflerle, özellik tanımları ilk harfi büyük olacak şekilde tanımlanabilir. Bilgi gizleme bölümünde bu konu ile ilgili daha detaylı bilgi sunulacaktır.  
  
Herhangi bir işlevin özellik veya metot olarak gerçekleştiriminin yapılması ayrımında, genel kural olarak sınıfa ait bir nitelik gereksinimi varsa özellik olarak tanımlanmalı, sınıfın yapması gereken bir işlem varsa metot olarak tanımlanmalıdır.

Özellikler içerisindeki kod bloğu tamamlanmadan çıkış için C# return, VB.NET için Exit Property kullanılabilir.

return ve Exit, sırasıyla C# ve VB.NET için temel blok çıkış kelimeleridir. return aynı zamanda her iki dil için değer döndürme kelimesidir.

Metotlar

Metotlar, içerisinde gerçekleştirim yapılabilecek, klasik programlama tanımı ile işlevlerdir.

Temel amacı işlevselliği sağlamak olan metotlar belirli bir işlemi gerçekleştirebilir ve aynı zamanda değer de döndürebilir. Değer döndürebilen metotlara fonksiyon adı verilmektedir.

Aşağıda örnek metotlar gösterilmiştir.

C#

**public** **class** Tablo {

// Değer döndürmeyen

**void** Ekle(**object** o) {

// Metot gerçekleştirimi

**if** (o == **null**)

**return**;

}

// Değer döndüren

**int** Averaj(**double** delta) {

// Metot gerçekleştirimi

**return** 0;

}

**void** Al(**ref** **string**[] sl) {

**string** [] l = **new** **string**[3];

l[0] = "A1";

l[0] = "A2";

l[0] = "A3";

sl = l;

}

}

VB.NET

**Public** **Class** Tablo

'Değer döndürmeyen

**Sub** Ekle(**ByVal** o **As** **Object**)

' Metot gerçekleştirimi

**If** o **Is** **Nothing** **Then**

**Return**

**End** **If**

**End** **Sub**

' Değer döndüren

**Function** Averaj(**ByVal** delta **As** Double) **As** **Integer**

' Metot gerçekleştirimi

**Return** 0

**End** **Function**

**Sub** Al(**ByRef** sl() **As** **String**)

**Dim** l(2) **As** **String**

l(0) = "A1"

l(0) = "A2"

l(0) = "A3"

sl = l

**End** **Sub**

**End** **Class**

Kod 68, Örnek Metotlar

Değer döndürmeyen metotlar C# için void, VB.NET için Sub rezerve kelimesi ile tanımlanmaktadır. C# için fonksiyon tanımı fonksiyonun döndüreceği tip ile başlamakta, VB.NET için Function rezerve kelimesi kullanılmaktadır.

C#, bir tanımın özellik veya metot olduğunu parantezleri kullanarak tanımaktadır. Metoda ait parametre bulunmuyorsa dahi parantez kullanılmalıdır.

**Metot Parametreleri**

Metotlar, çağrımlarda dışarıdan atanabilecek parametreler içerebilir. Yukarıdaki örnekte Ekle isimli metot object tipinde bir parametre içermektedir. Metodun içerdiği parametreler , (virgül) karakteri ile ayrılmalıdır.

Parametreler aynı zamanda niteleyiciler (ing. Modifiers) içerebilir. Parametre niteleyicilerinin temel işlevi metoda gönderilen ve geri değeri alınabilen parametrelerin ne şekilde yönetileceğini belirlemektir.

Aşağıda metot niteleyicileri ve açıklamaları gösterilmiştir.

| C# | VB.NET | Açıklama |
| --- | --- | --- |
| [[5]](#footnote-5) | ByVal | Parametrenin değer olarak metoda iletileceğini ifade eder. |
| ref | ByRef | Metodu çağıran tarafından parametreye değer atanması gerektiğini ve metot içerisinde değiştirilen parametre değerinin çağırana döndürüleceğini ifade eder.   Parametrenin referans olarak metoda iletileceğini ifade eder. |
| out | ByRef | Parametreye metot içerisinde mutlaka değer ataması yapılması gerektiğini ifade eder. [[6]](#footnote-6) |
| [[7]](#footnote-7) | Optional | Parametrenin isteğe bağlı olduğunu ifade eder. |
| params | ParamArray | Aynı tipte istenilen sayıda parametrenin iletilebileceğini ifade eder. |

Tablo 25, Metot Parametre Niteleyicileri ve Açıklamaları

*Değer Bazlı Parametreler*

Değer bazlı parametreler, iletilen parametre değerinin metoda değer bazlı iletimini ve metot içerisinde parametre değerlerinde yapılacak değişimin dışarıya yansımamasını sağlamaktadır.

C#

**class** Program {

**static** **int** Ekle(**int** i, **int** j) {

**int** sonuc = i + j;

i = j = 12000;

// çağıran tarafında yapılan değişiklik etkin olmayacaktır.

**return** sonuc;

}

**static** **void** Main(**string**[] args) {

**int** x = 5, y = 12;

Console.WriteLine(Ekle(x, y));

Console.WriteLine("x={0},y={1}", x, y);

}

}

VB.NET

**Class** Program

**Shared** **Function** Ekle(**ByVal** i **As** **Integer**, **ByVal** j **As** **Integer**) **As** **Integer**

**Dim** sonuc **As** **Integer**

sonuc = i + j

j = 12000

i = 12000

'çağıran tarafında yapılan değişiklik etkin olmayacaktır.

**Return** sonuc

**End** **Function**

**Shared** **Sub** Main(**ByVal** args() **As** **String**)

**Dim** y **As** **Integer** = 12

**Dim** x **As** **Integer** = 5

Console.WriteLine(Ekle(x,y))

Console..WriteLine("x={0},y={1}", x, y)

**End** **Sub**

**End** **Class**

Kod 69, Değer Bazlı Parametreler

Yukarıdaki örnek kod parçası çalıştırıldığında aşağıdaki çıktı üretilecektir.

17

x=5,y=12

Kod içerisindeki açıklamalardan da anlaşılacağı üzere, Ekle isimli metot içerisinde parametrelere atanan değerler, metot çağrımında kullanılan x ve y yerel değişkenlerinin değerini değiştirmemiştir.

*Referans Bazlı Parametreler*

Referans bazlı parametreler, iletilen parametre değerinin metot içerisinde değiştirilmesi durumunda çağrımda kullanılan parametrelerin de bu değişimden etkilenmesini sağlamaktadır.

Aşağıda referans parametre kullanım örnekleri gösterilmiştir.

C#

**static** **void** Artir(**ref** **int** x) {

x++;

}

**static** **void** Doldur(**out** **string** s, **out** **int** x) {

s = "Test";

x = 12;

}

**static** **void** Main(**string**[] args) {

**int** m, n = 1;

**string** s;

Doldur(**out** s, **out** m);

Artir(**ref** n);

Console.WriteLine("{0},{1},{2}", m, n, s);

}

VB.NET

**Shared** **Sub** Artir(**ByRef** x **As** **Integer**)

x = x + 1

**End** **Sub**

**Shared** **Sub** Doldur(**ByRef** s **As** **String**, **ByRef** x **As** **Integer**)

s = "Test"

x = 12

**End** **Sub**

**Shared** **Sub** Main(**ByVal** args() **As** **String**)

**Dim** n **As** **Integer** = 1

**Dim** m **As** **Integer**

**Dim** s **As** **String**

Doldur(s, m)

Artir(n)

Console.WriteLine("{0},{1},{2}", m, n, s)

**End** **Sub**

Kod 70, Referans Bazlı Parametreler

Kod örneği çıktısı aşağıda gösterilmiştir.

12,2,Test

out parametre niteleyicisinin temel kullanım amacı, ilk değeri atanmamış değişkenlerin metot içerisinde değer atamalarının yapılacağını derleyiciye iletmektir. ref, farklı olarak, ilk değer ataması yapılmış olan değişken değerlerinin metot içerisinde değiştirilebileceğini ifade etmektedir.  
  
C# kullanımında, çağrımda da out ve ref niteleyicilerinin bulunması zorunludur.

Referans parametre kullanarak metoda parametre iletimi ve iletilen parametre değerlerinin metot içerisinde belirlenmesi, farklı bir yaklaşımla metodun fonksiyon olarak gerçekleştirimi, değer almak için kullanılabilir.

C#

**static** **void** hesaplaA(**int** yuzde, **out** **int** sonuc) {

sonuc = yuzde \* 5;

}

**static** **int** hesaplaB(**int** yuzde) {

**return** yuzde \* 5;

}

**static** **void** Main(**string**[] args) {

**int** sonuc;

hesaplaA(25, **out** sonuc);

Console.WriteLine(sonuc.ToString());

Console.WriteLine(hesaplaB(25).ToString());

}

VB.NET

**Shared** **Sub** hesaplaA(**ByVal** yuzde **As** **Integer**, **ByRef** sonuc **As** **Integer**)

sonuc = yuzde \* 5

**End** **Sub**

**Shared** **Function** hesaplaB(**ByVal** yuzde **As** **Integer**) **As** **Integer**

**Return** yuzde \* 5

**End** **Function**

**Shared** **Sub** Main(**ByVal** args() **As** **String**)

**Dim** sonuc **As** **Integer**

hesaplaA(25, sonuc)

Console.WriteLine(sonuc.ToString)

Console.WriteLine(hesaplaB(25).ToString)

**End** **Sub**

Kod 71, Referans Bazlı Parametre veya Fonksiyon Kullanımı

Yukarıdaki örnekte hesaplaA isimli metot sonuc parametre değerine gerekli atamayı yapmakta, hesaplaB isimli fonksiyon ise hesaplanan değeri döndürmektedir. Genel yaklaşım olarak bu gibi durumlarda olabildiğince fonksiyon kullanımı önerilmektedir.

*İsteğe Bağlı Parametreler*

İsteğe bağlı parametreler (ing. Optional Parameters), çağrımda parametrenin bulunmasını zorunlu olmaktan çıkarmaktadır. Metot parametre listesinde, isteğe bağlı parametre kullandıktan sonra zorunlu parametre kullanılamamaktadır.

Aşağıdaki VB.NET kodunda isteğe bağlı parametre kullanımı gösterilmiştir.

**Sub** Hesapla(**ByVal** sayi **As Integer, Optional ByVal** delta **As Integer = 2**)

**End Sub**'Her iki kullanım da geçerlidir  
  
Hesapla(4)  
Hesapla(4, 3)

Kod 72, İsteğe Bağlı Parametre Kullanımı (VB.NET)

Benzer işlem C# için Metot Yükleme ile yapılabilir.

*Dizisel Parametreler*

Dizisel parametreler yardımıyla aynı tipte istenilen sayıda parametre metoda iletilebilir.

C#

**static** **int** Topla(**params** **int** [] sayilar) {

**int** sonuc = 0;

**foreach** (**int** sayi **in** sayilar)

sonuc += sayi;

**return** sonuc;

}

**static** **void** Main(**string**[] args) {

**int** toplam;

toplam = Topla(3, 6, 7, 1);

Console.WriteLine(toplam.ToString());

}

VB.NET

**Shared** **Function** Topla(**ByVal** **ParamArray** sayilar() **As** **Integer**) **As** **Integer**

**Dim** sonuc **As** **Integer** = 0

**For** **Each** sayi **As** **Integer** **In** sayilar

sonuc = sonuc + sayi

**Next**

**Return** sonuc

**End** **Function**

**Shared** **Sub** Main(**ByVal** args() **As** **String**)

**Dim** toplam **As** **Integer**

toplam = Topla(3, 6, 7, 1)

Console.WriteLine(toplam.ToString())

**End** **Sub**

Kod 73, Dizisel Parametre Kullanımı

Yukarıdaki örnekte Topla isimli fonksiyona istenilen sayıda int tipinde parametre iletilebilir. Her bir parametre değerine foreach döngüsü ile ulaşmak mümkündür.

*Metot Yükleme*

Metot yükleme (ing. Method Overloading), aynı sınıf içerisinde isimleri birbirleri ile aynı olmasına rağmen parametre listesi değişebilen metotlara verilen tanımdır.

Aşağıda metot yükleme ile ilgili örnek gösterilmiştir.

C#

**int** Hesapla(**int** sayi, **int** deltaMin, **int** deltaMax) {

**return** sayi \* deltaMin \* deltaMax;

}

**int** Hesapla(**int** sayi, **int** deltaMin) {

**return** Hesapla(sayi, deltaMin, 100);

}

**static** **int** Hesapla(**int** sayi) {

**return** Hesapla(sayi, 1);

}

VB.NET

**Overloads** **Function** Hesapla(**ByVal** sayi **As** **Integer**, **ByVal** deltaMin **As** **Integer**, \_

**ByVal** deltaMax **As** **Integer**) **As** **Integer**

**Return** sayi \* deltaMin \* deltaMax

**End** **Function**

**Overloads** **Function** Hesapla(**ByVal** sayi **As** **Integer**, \_  
 **ByVal** deltaMin **As** **Integer**) **As** **Integer**

**Return** Hesapla(sayi, deltaMin, 100)

**End** **Function**

**Overloads** **Function** Hesapla(**ByVal** sayi **As** **Integer**) **As** **Integer**

**Return** Hesapla(sayi, 1)

**End** **Function**

Kod 74, Metot Yükleme Örneği

Örnekten de anlaşılabileceği üzere Hesapla isminde farklı parametre sayıları olan üç adet fonksiyon tanımı yapılmıştır.

VB.NET için metot yükleme işleminde Overloads rezerve kelimesinin kullanımı zorunludur.

Referans ve Değer Tipler

Kitabın daha önceki konularında da bahsedildiği üzere .NET tip sistemi temel olarak referans ve değer tipler üzerine kuruludur.

Her iki tipin CLR tarafından yönetimi farklılıklar göstermektedir. Bu bölümde nesne oluşturma çerçevesinde referans ve değer tipler hakkında bilgiler verilecektir.

| C# | VB.NET |
| --- | --- |
| **public** **struct** Nokta {  **public** **int** X;  **public** **int** Y;  **public** Nokta(**int** x, **int** y) {  **this**.X = x;  **this**.Y = y;  }  **public** **override** **string** ToString() {  **return** **string**.Format  ("{0},{1}", X, Y);  }  } | **Public** **Structure** Nokta  **Public** X **As** **Integer**  **Public** Y **As** **Integer**  **Public** **Sub** **New**(**ByVal** x **As** **Integer**,   **ByVal** y **As** **Integer**)  **Me**.X = x  **Me**.Y = y  **End** **Sub**  **Public** **Overrides** **Function**  ToString() **As** **String**  **Return**  **String**.Format("{0},{1}", X, Y)  **End** **Function**  **End** **Structure** |

Kod 75, Yapı Örneği

Bilindiği üzere yapı tipleri değer tipleri, nesneler ise referans tiplerdir. Yukarıdaki örnekte Nokta isimli yapı tip tanımı yapılmıştır. Aynı işlev benzer şekilde sınıf tanımı ile de aşağıdaki şekilde yapılabilir.

| C# | VB.NET |
| --- | --- |
| **public** **class** NoktaN {  **public** **int** X;  **public** **int** Y;  **public** NoktaN(**int** x, **int** y) {  **this**.X = x;  **this**.Y = y;  }  **public** **override** **string** ToString() {  **return** **string**.Format  ("{0},{1}", X, Y);  }  } | **Public** **Class** NoktaN  **Public** X **As** **Integer**  **Public** Y **As** **Integer**  **Public** **Sub** **New**(**ByVal** x **As** **Integer**,  **ByVal** y **As** **Integer**)  **Me**.X = x  **Me**.Y = y  **End** **Sub**  **Public** **Overrides** **Function**  ToString() **As** **String**  **Return**  **String**.Format("{0},{1}", X, Y)  **End** **Function**  **End** **Class** |

Kod 76, Yapı Benzeri Sınıf Örneği

Kullanımları değerlendirildiğinde aşağıdaki kod parçası yapı ve sınıf tipleri için geçerlidir.

| C# | VB.NET |
| --- | --- |
| Nokta n = **new** Nokta(12, 6); NoktaN nn = **new** NoktaN(12, 6); | **Dim n As** Nokta **= New** Nokta(12,6) **Dim nn As** NoktaN **= New** NoktaN(12,6) |

Kod 77, Yapı ve Nesne Tanım Örneği

Nesne Oluşturma

new rezerve kelimesi nesne oluşturmak için kullanılmaktadır.

Aşağıda nesne tanımlama ve oluşturma ile ilgili bazı örnekler verilmiştir.

| C# | VB.NET |
| --- | --- |
| ArrayList list = **new** ArrayList();  list.Add(12);  **object** o;  o = **new** ArrayList();  o = **new** SortedList();  **int** j; | **Dim** list **As** ArrayList  list = **New** ArrayList()  list.Add(12)  **Dim** o **As** **Object**  o = **New** ArrayList()  o = **New** SortedList()  **Dim** j **As** **Integer** |

Kod 78, Nesne Oluşturma ve Değişken Tanımlama

Yukarıdaki örnek incelendiğinde, list isminde bir değişken tanımlanmış, tanımlama ile birlikte de ArrayList sınıfına ait bir nesne oluşturulmuştur. Benzer şekilde o isminde, tipi object sınıfı olan bir değişken tanımlanmıştır. o isimli değişkene daha sonra sırasıyla oluşturulan ArrayList ve SortedList nesneleri atanmıştır. Son olarak j isminde int tipinde bir değişken tanımlaması yapılmıştır.

Değer tipler için değişkenin tanımlanması ile değişken kullanıma hazır olmaktadır. Örneğin

**Dim** I **As Integer**  
I = 12

Örnekte I ismindeki değişken tanımlanmış ve bir sonraki satırda kullanılmıştır. Aşağıdaki örnekte ArrayList sınıfında bir değişken tanımlanmıştır

**Dim** list **As** ArrayList  
list.Add(12)

ve new kelimesi ile nesne oluşturulmadan kullanılmıştır. Bu durumda çalışma anında "Object reference not set to an instance of an object." hatası alınacaktır.

Değer tiplerin tanımlanması aşamasında değer tipin boyutuna göre otomatik olarak gerekli hafıza ayrımı derleyici tarafından yapılmaktadır. Böylece değer tipe atama yapıldığında veya değeri okunmak istendiğinde, ayrılan hafıza hücresi içerisinden değer okunabilmekte ve ilgili hafıza hücresi içerisine değer yazılabilmektedir. Yukarıdaki örnekte de tanımlama aşamasında 4byte boyutunda hafıza ayrımı otomatik olarak yapılmıştır.

Şekil 11, Değer Tipler Hafıza Yönetimi

Nesne tanımında ise derleyici tarafından tanım aşamasında yapılan sadece işaretçi tipinde bir değeri içerisinde saklayabilecek hafıza bölgesini ayırmaktır. Çalışma anında ne zaman new kelimesi ile nesne oluşturulursa, oluşturulan nesnenin kapsayacağı alan kadar hafıza ayrılmakta ve ayrılan hafıza adresi değişkene atanmakta, nesne oluşturma işlemleri başlamaktadır.

Şekil 12, Referans Tipler Hafıza Yönetimi

Nesne oluşturma ve dolayısıyla hafıza ayrımı işlemi çalışma anında yapıldığı için, nesne hiyerarşisi içerisinde en temel olan object tipindeki bir değişkene her türlü nesneyi atamak mümkündür. Polimorfizm kavramı içerisinde olan bu konu hakkında kitabın ilerleyen bölümlerinde daha detaylı bilgi verilecektir.

null, (VB.NET için nothing) nesne tanımında kullanılan değişkenin geçerli bir hafıza adresi içerip içermediği kontrolü için kullanılabilir. Örneğin  
  
if (o == null) o = new object()  
If o Is Nothing Then  
 o = new Object()  
End If  
  
null olan (veya geçersiz bilgi içeren) bir nesne tanımı, örneğin, aşağıdaki şekilde kullanıldığında   
  
ArrayList list;  
list.Add(12);  
  
Çalışma anında "Object reference not set to an instance of an object." hatası alınmasının sebebi, henüz geçerli bir hafıza adresini göstermeyen (nesne oluşturulmamış olduğu için) list nesne tanımına . operatörü ile giriş yapılmaya çalışılmasından kaynaklanmaktadır.  
  
Yukarıda aktarılan hafıza ayırma ve nesne oluşturma mantığı çerçevesinde nesne, 'sınıf örneği' olarak da adlandırılmaktadır.

Aşağıdaki tabloda temel kavramlar özetlenmiştir.

| Kavram | C# | VB.NET |
| --- | --- | --- |
| Sınıf Tanımı | public class {  …  } | Public Class  End Class |
| Nesne Tanımı | object o;  ArrayList list; | Dim o As Object;  Dim list As ArrayList; |
| Nesne Oluşturma | o = new object(); | o = New Object() |

Tablo 26, Temel Sınıf ve Nesne Kavramları Açıklamaları

Nesne oluşturma konusu .NET içerisinde programlama yaparken tüm detayları ile bilinmesi ve teorik düzeyde anlaşılmış olması gereken bir konudur. Nesne tanımı ile nesnenin oluşturulmayacağı, new kelimesi ile oluşturulduğu ve yukarıda anlatılan hafıza yönetiminin değer ve referans tipler için farklı olduğu unutulmamalıdır.

. (nokta) operatörü ile arka planda gerçekleşen işlem, nesne üye girişi yapılarak nesne için ayrılan hafıza alanına erişimdir. Aşağıdaki kod örnekleri incelendiğinde;

| C# | VB.NET |
| --- | --- |
| **void** TestStruct(Nokta p) {  n.X = n.Y = 90;  }  **void** TestClass(NoktaN p) {  p.X = p.Y = 90;  }  Nokta n;  n.X = 12;  n.Y = 12;  NoktaN nn = **new** NoktaN(12, 12);  TestStruct(n);  TestClass(nn);  Console.WriteLine(n.ToString());  Console.WriteLine(nn.ToString()); | **Sub** TestStruct(**ByVal** p **As** Nokta)  p.Y = 90  p.X = 90  **End** **Sub**  **Sub** TestClass(**ByVal** p **As** NoktaN)  p.Y = 90  p.X = 90  **End** **Sub**  **Dim** n **As** Nokta  n.X = 12  n.Y = 12  **Dim** nn **As** NoktaN  nn = **New** NoktaN(12, 12)  TestStruct(n)  TestClass(nn)  Console.WriteLine(n.ToString())  Console.WriteLine(nn.ToString()) |

Kod 79, Nokta Operatörü ve Değer Bazlı Parametreler

Daha önce sırasıyla yapı ve sınıf olarak tanımlanan Nokta ve NoktaN isimli sınıfların kullanıldığı örneğin çıktısı aşağıdadır.

12,12

90,90

Değer bazlı parametrelere metot içerisinde yapılan atamalar çağrımda kullanılan değişkenlerin değerini etkilememiştir. Bunun sebebi TestStruct metodu içerisindeki p isimli parametrenin, çağrımda kullanılan n isimli değişkenin bir kopyası olmasıdır.

Referans tiplerde de genel kural geçerlidir. Çağrımda kullanılan nn isimli değişkenin bir kopyası TestClass içerisinde p parametresidir. Buna rağmen . operatörü ile gerçek nesne referansı yapıldığı için, nesne alanları değiştirilmiş, böylece aynı nesne üzerinde değişiklik yapıldığı için çıktı 90, 90 şeklinde olmuştur.

Yapı tipleri için de new kelimesi kullanılabilir. Bu durum sadece yapının belirli ilk değerlerle tanımlanabilmesini sağlamaktadır. Temel amacı nesne oluşturma olan new kelimesinin bu şekilde kullanımı ile referans tipler karıştırılmamalıdır.

Referans ve Değer Tip Dönüşümü

Bilindiği gibi .NET içerisinde kullanılan her yapı System.Object sınıfını ata sınıf olarak kabul etmektedir.

Tip dönüşümü yardımıyla farklı tiplerin birbirlerine dönüştürülebileceği hakkında kitabın önceki bölümlerinde bilgi verilmişti. Aşağıdaki örnekte benzer durum gösterilmiştir.

| C# | VB.NET |
| --- | --- |
| **int** i = 12;  // Kutulama  **object** o = i;  // Kutudan çıkarma  **int** j = (**int**)o;  i = 5;  j = 6;  Console.WriteLine(i);  Console.WriteLine(o);  Console.WriteLine(j); | **Dim** i **As** **Integer** = 12  'Kutulama  **Dim** o **As** **Object** = i  'Kutudan çıkarma  **Dim** j **As** **Integer**  j = **CType**(o, **Integer**)  i = 5  j = 6  Console.WriteLine(i)  Console.WriteLine(o)  Console.WriteLine(j) |

Kod 80, Kutulama ve Kutudan Çıkarma

**Kutulama**

Kutulama (ing. Boxing), değer tipin referans tipe dönüşümü ve değer tipin System.Object içerisinde depolanmasını ifade etmektedir.

Yukarıdaki örnekte değer tip değişkeni olan i, referans tip değişkeni olan o içerisine kutulanmıştır. Kutulama işleminde CLR tarafından yapılan işlem, kutulanmak istenilen değer tipi boyutunda otomatik olarak hafıza ayrımı yapmak ve değer tip içeriğini bu hafıza alanına kopyalamaktır.

**Kutudan Çıkarma**

Kutudan çıkarma (ing. UnBoxing) işlemi kutulama işleminin tersidir. Kutudan çıkarma işleminde CLR tarafından yapılan, referans değer içerisinde depolanan değerin döndürülmesidir.

Yukarıdaki örnek çıktısı aşağıdaki gibi olacaktır.

5  
12  
6

Kutulama işleminde dikkat edilmesi gereken kural yeni bir hafıza ayrımının yapıldığıdır.

Değer tipler stack üzerinde referans tipler heap üzerinde saklanmaktadır.

System.Object Sınıfı

System.Object, .NET ata sınıfıdır. Tüm yapılar (değer tipler, diziler, sayılabilir tipler, vb.) bu sınıfı ata sınıf olarak kabul etmektedir.

Aşağıda sınıf üye bilgileri listelenmiştir.

| Üye Adı | Açıklama |
| --- | --- |
| GetType | Nesneye ait tip bilgisini döndürür. |
| Equals | Karşılaştırılan nesne ile aynı hafıza adresinde olmaları durumunda True döndürür. |
| GetHashCode | Hafıza içerisindeki nesneyi belirtebilecek olan tamsayı değer döndürür. |
| ToString | Belirtilen nesnenin karakter dizi görünümünü döndürür. |

Tablo 27, System.Object Sınıfı Temel Üye Bilgileri

Sınıf Yapıcıları

Sınıf yapıcıları (ing. Class Constructors), nesneyi oluşturan özel metotlardır. Bir başka anlatımla, yapıcılar, nesnenin oluşturulması aşamasında otomatik olarak çalışan metotlardır.

Aşağıdaki kod örneği incelendiğinde

ArrayList list = new ArrayList(12);

ArrayList isimli sınıfın yapıcısı otomatik olarak çalıştırılmaktadır.

| C# | VB.NET |
| --- | --- |
| **public** **class** Form {  **private** **int** x;  **public** **int** X {  **get** { **return** x; }  **set** { x = value; }  }  **private** **int** y;  **public** **int** Y {  **get** { **return** y; }  **set** {y = value; }  }  **public** Form(**int** x, **int** y) {  **this**.x = x;  **this**.y = y;  }  **public** Form(): **this**(0, 0) {  }  } | **Public** **Class** Form  **Private** \_x **As** **Integer**  **Private** \_y **As** **Integer**  **Public** **Sub** **New**(**ByVal** x **As** **Integer**,   **ByVal** y **As** **Integer**)  **Me**.\_x = x  **Me**.\_y = y  **End** **Sub**  **Public** **Sub** **New**()  **Me**.**New**(0, 0)  **End** **Sub**  **Public** **Property** X() **As** **Integer**  **Get**  **Return** X  **End** **Get**  **Set**(**ByVal** value **As** **Integer**)  \_x = Value  **End** **Set**  **End** **Property**  **Public** **Property** Y() **As** **Integer**  **Get**  **Return** Y  **End** **Get**  **Set**(**ByVal** value **As** **Integer**)  \_y = Value  **End** **Set**  **End** **Property**  **End** **Class** |

Kod 81, Sınıf Yapıcı Örnekleri

Yukarıdaki örnekte Form isimli sınıfa ait iki adet yapıcı tanımlanmıştır. C# için yapıcı tanımı, sınıf ismi ile aynı isme sahip metot tanımlamaktır. VB.NET için yapıcı tanımlamada New rezerve kelimesi kullanılmalıdır.

Yukarıdaki C# kod örneğinde sınıf alanları olan x ve y ile yapıcı parametreleri olan x ve y yapıcı metot içerisinde this rezerve kelimesi ile ayrılmıştır. this, (VB.NET için Me), sınıf içerisinden nesne referansına ulaşmak için kullanılmaktadır. Konuyla ilgili detaylı bilgi kitabın ilerleyen bölümlerinde verilecektir.  
  
Genel tasarım yaklaşımı olarak benzer durumlarda, tüm metot parametreleri isimlendirmeleri ve alanlara ulaşım için bu yöntem uygulanabilir..

Varsayılan Yapıcı

Bir sınıf içerisinde herhangi bir yapıcı tanımlanmazsa parametre içermeyen varsayılan yapıcı otomatik olarak tanımlanacaktır.

Örneğin

| C# | VB.NET |
| --- | --- |
| **public class** Test {  }  // Geçerli kullanım Test t = **new** Test(); | **Public Class** Test  **End Class**  ' Geçerli kullanım **Dim** t **As** Test = **new** Test() |

Kod 82, Varsayılan Yapıcı Örneği

Buna rağmen herhangi bir parametre içeren bir yapıcı tanımı yapıldığında varsayılan yapıcı geçersiz olmaktadır. Örneğin

| C# | VB.NET |
| --- | --- |
| **public class** Test {  **public** Test(**int** x) {  }  }  // Geçersiz kullanım Test t = **new** Test(); | **Public Class** Test  **Public Sub New**(**ByVal** x **As** Integer)  **End Sub**  **End Class**  ' Geçersiz kullanım **Dim** t **As** Test = **new** Test() |

Kod 83, Varsayılan Yapıcının Örtülmesi

Birden Fazla Yapıcı Kullanımı

Bir sınıf içerisinde birden fazla yapıcı tanımlanabilir. Örneğin yukarıdaki Form isimli sınıf örneğinde, iki adet parametre alan ve herhangi bir parametre almayan yapıcılar tanımlanmıştır.

Genel kural olmamakla birlikte, birden fazla yapıcı kullanımında, kaynak kodun bakımını da daha kolay yapabilmek amacıyla, kod tek yapıcı içerisinde toplanabilir ve diğer yapıcılar tarafından ilgili yapıcı çağrılabilir. Form isimli sınıf içerisinde tanımlanan parametresiz yapıcı, this ve New (sırasıyla C# ve VB.NET) rezerve kelimeleri yardımıyla, iki adet parametre alan yapıcıyı varsayılan değerlerle çağırmaktadır. Böylece, sınıf alanlarının değer atamaları sadece bir yapıcı tarafından yapılmaktadır. Aynı işlemin aşağıdaki şekilde yapılması önerilmemektedir.

| C# | VB.NET |
| --- | --- |
| **public** **class** Form {  **public** Form(**int** x, **int** y) {  **this**.x = x;  **this**.y = y;  }  **public** Form() {  **this**.x = 0;  **this**.y = 0;  }  } | **Public** **Class** Form  **Public** **Sub** **New**(**ByVal** x **As** **Integer**,   **ByVal** y **As** **Integer**)  **Me**.\_x = x  **Me**.\_y = y  **End** **Sub**  **Public** **Sub** **New**()  **Me**.\_x = 0  **Me**.\_y = 0  **End** **Sub**  **End** **Class** |

Kod 84, Birden Fazla Yapıcı Kullanım Örneği

Yapıcıları, nesne oluşturma işleminde otomatik olarak çalışan ve hafıza adresi döndüren metotlar olarak tanımlamak yanlış olmayacaktır. Aşağıdaki her iki örnek kod incelendiğinde  
  
Test t = new Test();  
int i = 12;  
  
atama işlemlerinin yapıldığı görülecektir. Atama işleminde değişkenin tipi ile atanan değerin aynı olması (veya uyumlu) değerlendirildiğinde new Test() ifadesinin Test tipinde bir değer döndürmesi beklenmektedir. Bu değeri gerçekte döndüren sınıfın yapıcısıdır.  
  
Not: Yapıcı, 'yapıcı metot' olarak da anılabilmektedir.

Statik Sınıf ve Özellikleri

Nesne tanımı aşağıda tekrar sunulmuştur.

Nesne, bir sınıfın oluşturulmuş şeklidir. (Sınıf örneğidir, ing. Instance of a class)

Tanım ve aktarılan bilgiler ışığında;

* Sınıf bir tip tanımıdır, nesne ise new rezerve kelimesi ile bu tipin yapıcı metodunun döndürdüğü değerdir.
* Nesne tanımı yapmak nesneyi kullanmak için yeterli değildir. Nesnenin kullanılabilmesi için nesnenin oluşturulmuş olması gerekmektedir.
* Nesne üyelerine giriş . operatörü ile yapılmaktadır. Bu operatörün kullanımı ile arka planda yapılan işlem, değişkenin işaret ettiği ve nesnenin saklandığı hafıza alanına giriştir.

Statik sınıf ve üyeler, işlevselliği nesneler yerine direkt sınıf ile sağlamayı mümkün kılmaktadır. Bazı durumlarda işlevselliğin farklı nesneler aracılığıyla sağlanması yerine sadece sınıf bazında veya her ikisi ile birlikte yapılması daha uygun olabilir.

Basit bir konsol uygulaması kaynak kodu aşağıda gösterilmiştir.

| C# | VB.NET |
| --- | --- |
| **class** Program {  **static** **void** Main(**string**[] args) {  }  } | **Class** Program  **Shared** **Sub** Main(**ByVal** args()   **As** **String**)  **End** **Sub**  **End** **Class** |

Tablo 28, Basit Konsol Uygulaması

Yukarıdaki kaynak kod derlendiğinde çalışabilir dosya üretilmekte ve her programın başlangıç metodu olan Main otomatik olarak çalışmaktadır.

Daha önce bir nesneyi oluşturmadan sadece tanım aracılığı ile üye girişi yapıldığında bunun çalışma anında hataya yol açacağı iletilmişti. Yukarıdaki örnekte ise herhangi bir nesne oluşturmadan Program isimli sınıfın Main metodun başarıyla çalışmaktadır.

Benzer şekilde aşağıdaki örnek irdelendiğinde

**string** s = **string**.Empty;

string sınıfına ait Empty isimli alana herhangi bir nesne oluşturmadan giriş yapılabildiği görülmektedir.

Farklı tasarım şablonlarının gerçekleştiriminde de avantajlar sağlayabilen statik kullanım C# için static, VB.NET için Shared rezerve kelimeleri ile yapılmaktadır.

Statik Sınıflar

Bir sınıfın statik olarak tanımlanması, sınıfın sadece statik üyelerinin olabilmesini zorunlu kılmakta ve sınıf tipinde nesne oluşturulamamaktadır.

**static** **class** Test {

**static** **void** Method() {

}

// Tanım geçersizdir

**void** GecersizMethod() {

}

}  
  
// Kullanım  
Test.Method()  
  
// Geçersiz nesne oluşturma, derleme anı hatası üretecektir.  
Test t = **new** Test();

Kod 85, Statik Sınıf Tanımı

Statik sınıf tanımı sadece C# tarafından desteklenmektedir. VB.NET için Module benzer işlevi sağlamaktadır.

Statik Üyeler

Temel olarak alanlar, özellikler ve metotlar (yapıcı da özel bir metot olarak değerlendirilebilir) olarak özetlenebilecek olan üyeler, gerek sınıf bazında (tip için) gerek nesne bazında kullanılabilir.

Statik üye – sınıf bazında üye – tanımı yapmak için üye tanımının static veya Shared (sırasıyla C# ve VB.NET) rezerve kelimesi ile donatılması yeterlidir.

C#

**class** NesneFabrikasi {

**private** **static** ArrayList list = **new** ArrayList();

**public** **static** **int** ToplamNesne {

**get** {

**return** list.Count;

}

}

**public** **static** **object** Uret() {

**object** o = **new** **object**();

list.Add(o);

**return** o;

}

**private** NesneFabrikasi() {

}

}

**static** **class** Program {

**static** **void** Main(**string**[] args) {

**for** (**int** i = 0; i < 5; i++) {

NesneFabrikasi.Uret();

}

Console.WriteLine(NesneFabrikasi.ToplamNesne);

}

}

VB.NET

**Class** NesneFabrikasi

**Private** **Shared** list **As** ArrayList = **New** ArrayList()

**Private** **Sub** **New**()

**End** **Sub**

**Public** **Shared** **ReadOnly** **Property** ToplamNesne() **As** **Integer**

**Get**

**Return** list.Count

**End** **Get**

**End** **Property**

**Public** **Shared** **Function** Uret() **As** **Object**

**Dim** o **As** **Object** = **New** **Object**()

list.Add(o)

**Return** o

**End** **Function**

**End** **Class**

**Shared** **Sub** Main(**ByVal** args() **As** **String**)

**For** i **As** **Integer** = 0 **To** 4

NesneFabrikasi.Uret()

**Next**

Console.WriteLine(NesneFabrikasi.ToplamNesne.ToString())

**End** **Sub**

**End** **Class**

Kod 86, Statik Üye Kullanımı

Yukarıdaki örnekte statik olarak tanımlanan alan ve üyeler yardımıyla nesne oluşturma işlemi sınıf tarafından yapılmakta ve üretilen nesneler de bir listede tutulmaktadır.

Yapıcının private ile donatılması dışarıdan görülememesini sağlamakta, böylece NesneFabrikasi tipinde bir nesne, sınıfın Uret() metodu dışında oluşturulamamaktadır Bu konuda kitabın ilerleyen bölümlerinde daha detaylı bilgi verilecektir..  
  
 Listelenen kaynak kodlar \Bölüm 5\Statics klasöründe bulunmaktadır.

Statik Yapıcılar

Statik yapıcılar, bir sınıfın ilk defa referans verilmesi ve kullanılması durumunda otomatik olarak çalıştırılan özel metotlardır.

Aşağıda örnek statik sınıf yapıcısı gösterilmiştir.

| C# | VB.NET |
| --- | --- |
| **Class** NesneFabrikasi {  **private** **static** **int** toplamNesne;  **static** NesneFabrikasi() {  toplamNesne = 0;  }  } | **Class** NesneFabrikasi  **Private Shared** toplamNesne **As Integer**  **Shared** **Sub** **New**()  toplamNesne = 0  **End** **Sub**  **End** **Class** |

Kod 87, Statik Yapıcılar

Kullanım Önerileri

Statik sınıf ve üyelerin kullanımında dikkat edilmesi önerilen konular aşağıda listelenmiştir.

* Sınıf tipinde tüm nesneleri ilgilendiren alan, özellik ve metotlar statik olarak tanımlanmalıdır. Örneğin string sınıfı içerisinde, tüm string tipinde nesneler için ortak olan boş karakter dizi değeri Empty alanı ile statik olarak tanımlanmıştır.
* Nesne oluşturma kontrolü sağlamak amacıyla statik üyeler tanımlayınız. Böylece sınıf, oluşturulan tüm nesneleri takip edebilme yeteneğine sahip olacaktır.
* Gerekli olmadığı durumlarda statik sınıf ve üye kullanmayınız. Genel tercih nesnelerle çalışma yönünde yapılmalıdır.

Statik üyeler, sınıf üyeleri olarak da adlandırılabilir. Adlandırmada geçerli olan mantık, bu üyelerin nesneye ait olmayıp sınıfa ait olmalarından dolayıdır. Kavram sayısını artırmamak için kitapta nesne üyeleri ve sınıf üyeleri şeklinde ayrım yapılmamış, sadece üye ve statik üye kavramları kullanılmıştır.

Nesneler – Sınıf Örnekleri

Daha önce verilen teorik bilgiler ışığında, bu bölümde nesneler hakkında özet bilgiler, kullanım detayları ve örnekler sunulacaktır.

Sınıf İçerisinden Nesneye Ulaşım

Bilindiği gibi nesnelerin bir diğer tanımı da sınıf örnekleridir. Bir sınıfa ait istenilen sayıda nesne oluşturmak mümkündür.

Şekil 13, Sınıf ve Nesne

Sınıf içerisinden nesneye erişim this ve me (sırasıyla C# ve VB.NET) rezerve kelimeleri ile yapılmaktadır.

Sınıf içerisinde üyelere erişim direkt üye adı ile yapılabilir. Buna rağmen bazı durumlarda, örneğin metot parametre ismi ile nesneye ait bir alan veya özellik isminin aynı olması durumunda, erişimde this (Me) kelimesi ön ek olarak kullanılabilir. this.x veya Me.x gibi.

Hafıza Yönetimi

Daha önceki konularda da bahsedildiği üzere nesne tanımlama ve nesne oluşturma birbirinden farklı kavramlardır.

ArrayList list;

Yukarıdaki list isimli nesne tanımı ile yapılan işlem sadece ArrayList isimli sınıfa ait bir nesne adresini saklayabilecek kadar (32Bit sistemlerde 4Byte, 64Bit sistemlerde 8Byte) hafıza ayrımı yapmaktır. Bu mantık çerçevesinde tüm sınıflar için yapılan nesne tanımlarında, hafıza ayrım boyutunun aynı olduğu söylenebilir.

New rezerve kelimesi ile nesne oluşturulduğunda, sınıfın sahip olduğu üye sayısı gibi ölçütlerle CLR tarafından yeterli hafıza ayrımı yapılmaktadır.

Burada değerlendirilmesi gereken önemli konulardan birisi, nesnelerin yaşam sürecidir. .NET, yönetilen bir platform sağladığı için daha alt seviyede işaretçi kullanımı ve hafıza yönetimi gibi işlemlere güvenlik dolayısıyla izin vermemekte, bu işlemleri kendisi yönetmektedir. Dolayısıyla, oluşturulan nesneler gerektiğinde otomatik olarak CLR tarafından temizlenmektedir.

Çöp toplayıcısı (ing. Garbage Collector - GC), oluşturulan nesnelere hiçbir referansın kalmadığından emin olmakta ve belirli aralıklarla otomatik olarak kullanılmayan nesneleri hafızadan temizlemekte, hafızayı boşaltmaktadır.

C#

**public** **class** RandomList {

**public** **string**[] Uret() {

[U1] ArrayList list = **new** ArrayList();

[U2] Random r = **new** Random();

**int** max = r.Next(1000);

**for** (**int** i = 0; i < max; i++)

list.Add(r.Next().ToString());

[U3] **return** (**string** [])list.ToArray(**typeof**(**string**));

}

}

**static** **class** Program {

**static** **void** Main() {

[M1] RandomList l = **new** RandomList();

[M2] **string** [] sl = l.Uret();

**foreach** (**string** s **in** sl)

Console.Write("{0} ", s);

}

}

VB.NET

**Public** **Class** RandomList

**Public** **Function** Uret() **As** **String**()

[U1] **Dim** list **As** ArrayList = **New** ArrayList()

[U2] **Dim** r **As** Random = **New** Random()

**Dim** max **As** **Integer** = r.Next(1000)

**For** i **As** **Integer** = 0 **To** max - 1

list.Add(r.Next().ToString())

**Next**

[U3] **Return** **CType**(list.ToArray(**GetType**(**String**)), **String**())

**End** **Function**

**End** **Class**

**Class** Program

**Shared** **Sub** Main()

[M1] **Dim** l **As** RandomList = **New** RandomList()

[M2] **Dim** sl() **As** **String** = l.Uret()

**For** **Each** s **As** **String** **In** sl

Console.Write("{0} ", s)

**Next**

**End** **Sub**

**End** **Class**

Kod 88, Nesne ve Hafıza Kullanım Örneği

Listelenen kaynak kodlar \Bölüm5\Randoms klasöründe bulunmaktadır.

Yukarıdaki örnekte rasgele boyutta ve içerikte dizi üretebilen sınıf tanımı yapılmıştır. Uygulamanın giriş metodu olan Main içerisinde sırasıyla

* l isminde bir nesne oluşturulmuştur. (M1)
* Nesnenin dizi döndüren Uret() metodu kullanılarak dizi sl isimli yerel değişkene atanmıştır. (M2)

Uret() metodu içerisinde sırasıyla

* list isminde ve ArrayList tipinde bir nesne oluşturulmuştur. (U1)
* r isminde ve Random tipinde bir nesne oluşturulmuştur. (U2)
* Listeye elemanlar eklenmiş ve ArrayList.ToArray metodu yardımıyla liste içeriği dizi olarak return kelimesi ile döndürülmüştür. (U3)

Adımlarda toplam üç adet nesne oluşturulmuştur. Yönetilmeyen bir ortamda, programcı tarafından oluşturulan nesnenin yine programcı tarafından yok edilmesi (ing. Destroy) gerekmektedir. Yönetilen .NET platformunda bu işlem nesneye herhangi bir referansın kalmamasının kontrolü ile otomatik olarak GC tarafından yapılmaktadır.

M2 adımı tamamlandıktan sonra, oluşturulan list ve r isimli nesnelere herhangi bir ulaşımın mümkün olmadığı görülmektedir. Uret() metodunun tamamlanması ile bu nesnelerin işlevi de tamamlanmış olmaktadır. Belirli zaman aralıkları ile çalışan GC, örneğin, diğer gereksiz nesneler ile birlikte bu nesneleri de hafızadan atmaktadır.

Referans tipler ile değer tipler için hafıza yönetimi farklılıklar göstermektedir. .NET platformunda kullanılan her yapının nesne olduğu göz önüne alındığında, Uret() metodu içerisinde kullanılan max değişkeni de bir nesnedir. Buna rağmen, int tipinin değer tip olması dolayısıyla, Uret() metodundan çıkıldığında max için ayrılmış hafıza da otomatik olarak boşaltılmakta, referans tiplerde olduğu gibi GC tarafından belirli aralıklarla temizleme işlemi değer tipler için yapılmamaktadır.

Değer tipler için yöntemin yukarıda aktarılan şekilde olabilmesini sağlayan, değer tiplerin heap içerisinde değil stack içerisinde saklanmasından kaynaklanmaktadır.

Tip Dönüşümü

Önceki konularda temel tipler için aktarılan tip dönüşüm bilgileri benzer şekilde nesneler için de geçerlidir.

Yukarıdaki kod örneğinde, ArrayList.ToArray metodu System.Array döndürmektedir. Farklı tipte eleman içerebilen dizilerin soyutlanmış hali olarak özetlenebilecek System.Array sınıfı, tip dönüştürme işlemi ile karakter dizi dizisi (ing. String Array) nesnesine dönüştürülmüştür.

Bir nesnenin hangi sınıfa ait olduğu is operatörü yardımıyla anlaşılabilir. Aşağıdaki örnekte iletilen nesne ile ilgili bilgiler yazılmaktadır.

C#

**static** **class** Program {

**static** **void** Bilgi(**object** o) {

**if** (o == **null**) {

Console.WriteLine("Nesne null");

**return**;

}

**if** (o **is** ArrayList)

Console.WriteLine("ArrayList");

**else** **if** (o **is** **int**)

Console.WriteLine("Tam Sayı");

**else** Console.WriteLine( o.GetType().Name);

}

**static** **void** Main(**string**[] args) {

**object**[] list = **new** **object**[] {**null**, 12, **new** ArrayList(), "abc"};

**foreach** (**object** o **in** list)

Bilgi(o);

}

}

VB.NET

**Class** Program

**Shared** **Sub** Bilgi(**ByVal** o **As** **Object**)

**If** o **Is** **Nothing** **Then**

Console.WriteLine("Nesne null")

**Return**

**End** **If**

**If** (**TypeOf** (o) **Is** ArrayList) **Then**

Console.WriteLine("ArrayList")

**ElseIf** **TypeOf** (o) **Is** **Integer** **Then**

Console.WriteLine("Tam Sayı")

**Else**

Console.WriteLine(o.**GetType**.Name)

**End** **If**

**End** **Sub**

**Shared** **Sub** Main(**ByVal** args() **As** **String**)

**Dim** list() **As** **Object** = **New** **Object**() {**Nothing**, 12, **New** ArrayList, "abc"}

**For** Each o **As** **Object** **In** list

Bilgi(o)

**Next**

**End** **Sub**

**End** **Class**

Kod 89, Nesne Tip İşlemleri Kod Örneği

Kod çıktısı aşağıda gösterilmiştir.

Nesne null

Tam Sayı

ArrayList

String

Nesnelerin Parametrelere İletimi

Önceki konularda metot parametre niteleyicileri hakkında bilgiler verilmişti. Konunun daha kolay anlaşılabilmesi için kullanım örnekleri değer tip ağırlıklı olarak sunulmuştu.

Genel kurallar geçerli olmakla birlikte, referans tiplerin parametrelere iletimi konusunda aşağıdaki örneği inceleyiniz.

C#

**class** Test {

**static** **void** A(**out** **object** o) {

o = 12;

}

**static** **void** B(**ref** **object** o) {

o = **new** StringBuilder();

}

**static** **void** C(**object** o) {

o = 12;

}

**static** **void** D(**object** o) {

**if** (o **is** StringBuilder)

((StringBuilder)o) .Append("Test String\t");

}

**static** **void** E(**ref** **object** o) {

**if** (o **is** StringBuilder)

((StringBuilder)o).Append("Test String\t");

}

}

**class** Program {

**static** **void** Bilgi(**object** o) {

Console.WriteLine("{0}\t{1}", o.GetType().Name, o.ToString());

}

**static** **void** Main(**string**[] args) {

**object** x;

Test.A(**out** x); Bilgi(x); Test.B(**ref** x); Bilgi(x); Test.C(x);

Bilgi(x); Test.D(x); Bilgi(x); Test.E(**ref** x); Bilgi(x);

}

}

VB.NET

**Class** Test

**Shared** **Sub** A(**ByRef** o **As** **Object**)

o = 12

**End** **Sub**

**Shared** **Sub** B(**ByRef** o **As** **Object**)

o = **New** StringBuilder()

**End** **Sub**

**Shared** **Sub** C(**ByVal** o **As** **Object**)

o = 12

**End** **Sub**

**Shared** **Sub** D(**ByVal** o **As** **Object**)

**If** **TypeOf** (o) **Is** StringBuilder **Then**

**CType**(o, StringBuilder) .Append("Test **String**" + vbTab)

**End** **If**

**End** **Sub**

**Shared** **Sub** E(**ByRef** o **As** **Object**)

**If** **TypeOf** (o) **Is** StringBuilder **Then**

**CType**(o, StringBuilder).Append("Test String" + vbTab)

**End** **If**

**End** **Sub**

**End** **Class**

**Class** Program

**Shared** **Sub** Bilgi(**ByVal** o **As** **Object**)

Console.WriteLine("{0}" + vbTab + "{1}", o.**GetType**.Name, o.ToString())

**End** **Sub**

**Shared** **Sub** Main(**ByVal** args() **As** **String**)

**Dim** x **As** **Object**

Test.A(x)

Bilgi(x)

Test.B(x)

Bilgi(x)

Test.C(x)

Bilgi(x)

Test.D(x)

Bilgi(x)

Test.E(x)

Bilgi(x)

**End** **Sub**

**End** **Class**

Kod 90, Referans Tiplerin Parametre Olarak Kullanımı

Listelenen kaynak kodlar \Bölüm5\Objects klasöründe bulunmaktadır.

Nesne kullanımı ve hafıza yönetimi konusunda farklı yaklaşımlar hakkında bilgiler sağlayan örnek kod çıktısı aşağıda gösterilmiştir.

Int32 12

StringBuilder

StringBuilder

StringBuilder Test String

StringBuilder Test String Test String

| Metot | Açıklama |
| --- | --- |
| A | Nesne referansı metoda iletildiği için referans üzerinde yapılan değişiklik çağrıma yansımıştır. Böylece x nesnesi bir tamsayı nesnesine (int) dönüşmüştür. |
| B | Nesne referansı metoda iletildiği için referans üzerinde yapılan değişiklik çağrıma yansımıştır. Böylece x nesnesi bir StringBuilder nesnesine dönüşmüştür. |
| C | Değer bazlı iletim yöntemi uygulandığı için x nesnesinde herhangi bir değişiklik olmamıştır. |
| D | Değer bazlı iletim yöntemi uygulanmasına rağmen . operatörü ile üye girişi yapıldığı için (o tarafından işaret edilen nesneye ulaşıldığı için), çağrılan metot gerçekte x nesnesinin metodu olmuştur. |
| E | Nesne referansı metoda iletilmiştir. Buna rağmen D metodundan farklı bir işlem gerçekleşmemiştir.  Burada arka planda yapılan işlem, A ve B metotlarından farklı değildir. Metot içerisinde direkt o değişkenine bir atama yapılmadığı için çağrımda kullanılan x etkilenmemiştir. |

Tablo 29, Referans Tip Yönetimi Örnek Kod Açıklamaları

Bir sonraki konuya geçmeden önce kısa bir tekrar ve çeşitli kodlama çalışmaları yapılması önerilmektedir.

Temel NYP Özellikleri

Bilindiği gibi NYP üç temel gereksinim ve kavram üzerine kuruludur.

* Bilgi Gizleme ve Koruma (ing. Information Hiding & Encapsulation).
* Miraslık (ing. Inheritance).
* Polimorfizm (ing. Polymorphism).

Bu bölümde üç kavram çerçevesinde .NET ile NYP konusunda daha ileri seviye bilgiler sunulacaktır.

Şu ana kadar oldukça önemli görülen 'sınıf ve nesne kavram ayrımı'nı daha net aktarabilmek için yapılan farklı tanımlamalar, bu bölümden itibaren tamamen nesneler üzerine kurulu olarak sürdürülecektir.

Bilgi Gizleme ve Koruma

Bilgi gizleme ve koruma, bir sınıfı veya bir nesnenin içsel bütünlüğünü korumak amacıyla nesne veya nesne üyelerinin istenilen miktarının dışarıya açılabilmesini sağlayan ve gerçekleştirim detaylarını dışarıdan izole eden kavrama verilen isimdir.

Genel kural olarak bilgi gizleme ve koruma, nesne veya nesne üye tanımlarına, farklı anlamlar ifade eden giriş niteleyicilerini (ing. Access Modifiers) ön ek olarak kullanmakla sağlanmaktadır.

Bilgi gizleme ve koruma ile hedeflenen temelde güvenliktir. Örneğin bir programcı dağıtmış olduğu Assmbly'nin farklı programcılar tarafından da kullanılmasını ancak sadece istediği sınıfları diğer programcıların görebilmesini ve kullanabilmesini isteyebilir. Benzer şekilde, bir nesnenin kritik bir metodunun sadece içsel kullanımla çağrılabilmesini (sadece nesnenin kendisi tarafından) ancak aynı metodun dışarıdan çağrılamaması ihtiyacı doğabilir.

Farklı kullanımlarda farklı anlamlar taşıyabilen giriş niteleyicileri hakkında özet bilgi aşağıda sunulmuştur.

| C# | VB.NET | Açıklama |
| --- | --- | --- |
| public | Public | Sınıf veya üye tamamıyla dışarıya açıktır. |
| private | Private | Sadece ilgili nesne veya sınıf tarafından kullanılabilirliği ifade etmektedir. |
| protected | Protected | Sadece ilgili sınıfı ata sınıf olarak kabul eden (miras alan) sınıflara açık olmayı ifade etmektedir. |
| internal | Friend | Sadece aynı Assmbly içerisinde bulunan nesnelere açık olmayı ifade etmektedir. |
| protected internal | Protected Friend | Miras alan sınıflar ve aynı Assembly içerisinde açık olmayı ifade etmektedir. |
| readonly | ReadOnly | Üyelere salt okunur nitelik katmak için kullanılmalıdır. |

Tablo 30, Giriş Niteleyici Açıklamaları

Sınıf Gizleme ve Koruma

Bilindiği gibi derlemeden sonra oluşturulan Assembly içerisinde, farklı ad alanlarının mantıksal gruplaması altında bulunan sınıf ve nesne tanımları ile IL kod gerçekleştirimleri bulunmaktadır.

Using ve Imports (sırasıyla C# ve VB.NET) ifadeleri ile gerek aynı Assembly içerisinde bulunan ad alanları gerek farklı Assembly içerisinde bulunan ad alanları referans olarak verilebilir.

Nesne gizleme ve koruma, temel olarak bir Assembly içerisinde bulunan nesnelerin farklı bir Assembly içerisinden de kullanılıp kullanılamayacağını belirtmek için destek sağlamaktadır.

Aşağıda sınıf giriş niteleyicileri ve açıklamaları listelenmiştir.

| C# | VB.NET | Açıklama |
| --- | --- | --- |
| public | Public | Sınıf tamamıyla dışarıya açıktır ve farklı Assembly'lerden referans verilerek kullanılabilir. |
| private | Private | Sadece iç içe sınıf kullanımında içteki sınıf için kullanılabilir. Kullanımı durumunda içte bulunan sınıf, sadece kapsayan sınıf tarafından görülebilir. |
| protected | Protected | Sadece iç içe sınıf kullanımında içteki sınıf için kullanılabilir. Kullanımı durumunda içte bulunan sınıf, gerek kapsayan sınıf, gerek bu sınıfı miras alan diğer sınıflar tarafından görülebilir. |
| internal | Friend | Kullanımı durumunda sınıf, sadece bulunduğu Assembly içerisinden görülebilir, farklı bir Assembly tarafından sınıf kullanılamaz ve görülemez. |
| protected internal | Protected Friend | Her iki niteleyicinin birleşimidir. |

Tablo 31, Sınıf Giriş Niteleyici Açıklamaları

Sınıf tanımında herhangi bir giriş niteleyicisi kullanılmadığı durumda ön tanımlı olarak internal (VB.NET için Friend) varsayılmaktadır.

Sınıf giriş niteleyicileri ile ilgili kullanım örnekleri aşağıda verilmiştir.

| C# | VB.NET |
| --- | --- |
| **class** A {  }  **public** **class** B {  **protected** **class** C {  }  **private** **class** D {  }  }  **internal** **static** **class** E {  } | **Class** A  **End** **Class**  **Public** **Class** B  **Protected** **Class** C  **End** **Class**  **Private** **Class** D  **End** **Class**  **End** **Class**  **Friend Class** E  **End** **Class** |

Kod 91, Sınıf Giriş Niteleyici Örnekleri

Sınıf giriş niteleyicilerinin doğru kullanımı güvenli yazılımlar elde etmek için dikkat edilmesi gereken bir konudur. Temel prensip olarak sınıfın farklı Assembly'ler tarafından kullanım ihtiyacı bulunmuyorsa sınıf tanımında herhangi bir giriş niteleyicisi kullanılmadan bu güvenlik sağlanabilir.

Üye Gizleme ve Koruma

Sınıf üyeleri, sınıflara benzer şekilde, aşağıda listelenen üye giriş niteleyicileri yardımıyla korunabilir.

| C# | VB.NET | Açıklama |
| --- | --- | --- |
| public | Public | Üye tamamıyla dışarıya açıktır ve farklı Assembly'lerden referans verilerek kullanılabilir. |
| private | Private | Üye sadece tanımlandığı sınıf içerisinde görülebilir ve kullanılabilir durumdadır. |
| protected | Protected | Üye sadece tanımlandığı sınıf ve bu sınıfı miras alan sınıf tarafından görülebilir ve kullanılabilir. |
| internal | Friend | Üye sadece sınıf tanımının bulunduğu Assmbly içerisinden görülebilir ve kullanılabilir. |
| protected internal | Protected Friend | Her iki niteleyicinin birleşimidir. |

Tablo 32, Üye Giriş Niteleyici Açıklamaları

Üye giriş niteleyicileri ile ilgili çeşitli örnekler aşağıda gösterilmiştir.

| C# | VB.NET |
| --- | --- |
| **class** Test {  **static readonly** **double** PI = 22 / 7;  **private** **int** x = 1;  **public** **int** X {  **get** {  **return** x;  }  }  **protected** **void** B() {  }  }  //Kullanım Örnekleri  Test t = **new** Test();  // Geçerli Kullanım  Console.WriteLine(t.X);  // Geçersiz Kullanım, Protected  t.B();  // Geçersiz Kullanım, Salt okunur  Test.PI = 12.0; | **Class** A  **Shared** **ReadOnly** PI **As** Double = 22 / 7  **Private** \_x **As** **Integer = 1**  **Public** **ReadOnly** **Property** X()   **As** **Integer**  **Get**  **Return** \_x  **End** **Get**  **End** **Property**  **Protected** **Sub** B()  **End** **Sub**  **End** **Class**  ' Kullanım Örnekleri  **Dim** t **As** Test = **New** Test  ' Geçerli Kullanım  Console.WriteLine(t.X)  ' Geçersiz Kullanım, Protected  t.B()  ' Geçersiz Kullanım, Salt okunur  Test.PI = 12.0; |

Kod 92, Üye Giriş Niteleyici Örnekleri

**Yapıcı Metot Koruma ve Gizleme**

Yapıcı metotlar da yukarıda aktarılan giriş niteleyicileri yardımıyla aynı kurallarla korunabilir. Örneğin yapıcı metodun private ile korunması durumunda ilgili sınıf dışarıdan görülebilir ancak sınıf örneği (nesne) oluşturulamaz. Benzer şekilde protected ile korunması durumunda sadece sınıfı miras alan sınıf tarafından sınıf örneği oluşturulabilir.

| C# | VB.NET |
| --- | --- |
| **class** Test {  **public** **void** A() {  }  **protected** Test(**int** x) {  }  }  **class** Test2 : Test {  **public** Test2(): **base**(12) {  }  }  // Kullanımlar  Test2 t2 = **new** Test2();  // Geçersiz kullanım, yapıcı metot  // protected ile donatılmıştır.  Test t = **new** Test();  // Geçerli kullanım  t2.A(); | **Class** Test  **Protected** **Sub** **New**(**ByVal** x **As** **Integer**)  **End** **Sub**  **Public** **Sub** A()  **End** **Sub**  **End** **Class**  **Class** Test2  **Inherits** Test  **Public** **Sub** **New**()  **MyBase**.**New**(12)  **End** **Sub**  **End** **Class**  'Kullanımlar  **Dim** t2 **As** Test2 = **New** Test2()  ' Geçersiz kullanım, yapıcı metot  ' protected ile donatılmıştır.  **Dim** t **As** Test = **New** Test()  ' Geçerli Kullanım  t2.A() |

Kod 93, Yapıcı Metot Giriş Niteleyici Örnekleri

Tüm üyelerin tanımlarında giriş niteleyicilerinin belirtilmesi önerilmektedir.

Miraslık

Miraslık, ata sınıf tarafından belirtilen miraslık kuralları çerçevesinde, bir sınıfın işlev ve üyelerinin bir başka sınıf tarafından kod gerçekleştirimini yinelemeden kullanılabilmesi ve genişletilebilmesidir.

Miraslık ile hedeflenen, kodun tekrar kullanılabilirlik düzeyini sınıf hiyerarşisi çerçevesinde yükseltebilmektedir.

Şekil 14, Miraslık ve Ata Sınıf

Yukarıdaki şekilde kavramsal olarak gösterilen sınıf hiyerarşisinin C# ve VB.NET ile gerçekleştirimi aşağıdadır.

| C# | VB.NET |
| --- | --- |
| **class** TemelKullanıcı {  }  **class** YetkiliKullanıcı :  TemelKullanıcı {  }  **class** YöneticiKullanıcı :  YetkiliKullanıcı {  }  **class** Seçmen :  YetkiliKullanıcı {  } | **Class** TemelKullanıcı  **End** **Class**  **Class** YetkiliKullanıcı  **Inherits** TemelKullanıcı  **End** **Class**  **Class** YöneticiKullanıcı  **Inherits** YetkiliKullanıcı  **End** **Class**  **Class** Seçmen  **Inherits** YetkiliKullanıcı  **End** **Class** |

Kod 94, Miraslık Örnekleri

Bilindiği gibi belirtilsin veya belirtilmesin tüm sınıflar System.Object sınıfını miras almaktadır. Yukarıdaki örnekte de TemelKullanıcı sınıfı otomatik olarak System.Object sınıfını ata sınıf olarak kabul etmektedir.

C# için sınıf adı ile ata sınıf arasına : karakteri eklemek miraslık tanımını sağlamaktadır. VB.NET için rezerve bir kelime olan Inherits miraslık tanımı için kullanılmaktadır.

Sınıf Hiyerarşisi ve Soyutlama

Uygulama yazılımları geliştirmeden önce sınıfların ve sınıf hiyerarşisinin belirlenmesi büyük önem taşımaktadır. Sınıf hiyerarşisinin doğru kurgulanması ve tasarlanması bir çok durumda daha az kaynak kod geliştirerek, daha az hata olasılığı ile aynı işlevin gerçekleştirimini sağlamaktadır.

Sınıf hiyerarşisinin belirlenmesinde soyutlama kavramı önemlidir. Soyutlamanın tasarım aşamasında gerektiği düzeyde yapılması ve tasarlanan sınıfların gerçekleştiriminin yapılması temel yöntem olarak belirlenmelidir.

Yukarıda gösterilen örnek için aşağıdaki gereksinim ve bilgiler geçerlidir.

Sistemde somut olarak Seçmen ve Yönetici Kullanıcı rollerine sahip kullanıcılar bulunmaktadır. Aynı zamanda tasarımı yapılacak sistem ziyaretçi kullanıcılara da bilgi amaçlı hizmet verecektir.  
  
Temel Kullanıcı sınıfı, rolü ve niteliği ne olursa olsun herhangi bir kullanıcıyı belirtmek amacıyla kullanılacaktır. Bu sınıf, aynı zamanda, tüm sistem kullanıcılarının ortak özelliklerini yöneten sınıf olarak değerlendirilmektedir.  
  
Seçmen ve Yönetici Kullanıcı rolüne sahip olmak için kullanıcıların sisteme giriş yapmış olması gerekmektedir. Yetkili kullanıcı sınıfı , rolü ne olursa olsun sisteme giriş yapmış kullanıcıyı ifade etmektedir. Böylece sisteme giriş, yetki kontrolü gibi özellik ve işlevler bu sınıfta toplanmıştır.

İki derece soyutlamanın yapıldığı örnekte, ortak özellik ve işlevler ata sınıflara yüklenmiş, aynı işlemlerin farklı sınıflarda tekrarlanmasının önüne geçilmiştir.

Soyut Sınıf ve Üyeler

Soyut sınıflar, sınıfa ait nesne oluşturulması mümkün olmayan sınıflardır. Soyut sınıf tanımı abstract ve MustInherit (sırasıyla C# ve VB.NET) rezerve kelimeleri ile yapılmaktadır.

| C# | VB.NET |
| --- | --- |
| // abstract ile sınıfın donatılması // sınıfın soyut sınıf olmasını // sağlamaktadır. **abstract class** TemelKullanıcı {  }  **abstract** **class** YetkiliKullanıcı :  TemelKullanıcı {  }  // Aşağıdaki kullanım geçersizdir // ve derleme hatasına yol // açacaktır.  TemelKullanıcı t;  t = **new** TemelKullanıcı(); | ' MustInherit ile sınıfın  ' donatılması sınıfın soyut  ' sınıf olmasını sağlamaktadır. **MustInherit Class** TemelKullanıcı  **End** **Class**  **MustInherit Class** YetkiliKullanıcı  **Inherits** TemelKullanıcı  **End** **Class**  ' Aşağıdaki kullanım geçersizdir ' ve derleme hatasına yol ' açacaktır.  **Dim** t **As** TemelKullanıcı  t= **New** TemelKullanıcı() |

Kod 95, Soyut Sınıf Tanımı

Soyut sınıfların temel kullanım amacı, soyutlama çerçevesinde hiyerarşiyi belirlemek ve pratikte kullanım için miras alınmayı zorunlu kılmaktadır. Yukarıdaki örnekte, TemelKullanıcı ve YetkiliKullanıcı sınıfları gerçek hayatta bulunmayan tiplerdir. Buna rağmen tüm kullanıcıların özellikleri ve işlevlerinin gruplanması amacıyla soyutlama çerçevesinde bu sınıfların tasarımı yapılmıştır. Böyle bir durumda her iki sınıfı da soyut olarak tanımlamak, böylece bu sınıflara ait nesne oluşturabilmenin önüne geçmek anlamlıdır.

Sınıflar gibi sınıfa ait özellik ve metotları da soyut olarak tanımlamak mümkündür. Soyut üyelerin kod gerçekleştirimi bulunmamaktadır ve bir üyenin soyut olarak tanımlanabilmesi için sınıfın da soyut olarak tanımlanması zorunludur.

Üyelerin soyut olarak tanımlanmasındaki en önemli etken, soyut tanımlanan üyelerin ilgili sınıfı miras alan sınıflar tarafından mutlaka kod gerçekleştiriminin yapılması gerektiğini belirtmektedir.

Yukarıdaki örnek değerlendirildiğinde, YetkiliKullanıcı sınıfını miras alan tüm sınıfların yetki düzeyini tamsayı olarak ifade eden bir özellik ile kullanıcının bir işleme hakkı olup olmadığını belirten bir metoda sahip olmaları beklenebilir. Belirtilen iki üyeyi soyut olarak tanımlayarak, YetkiliKullanıcı sınıfını miras alan tüm sınıfların bu üyelere sahip olmaları ve üyelerin kod gerçekleştirimini yapmaları zorunluluğu sağlanabilir.

C#

**abstract** **class** YetkiliKullanıcı : TemelKullanıcı {

// Salt okunur ve soyut olarak olarak tanımlanan bu özellik

// ile YetkiliKullanıcı sınıfını miras alan her sınıfın bu

// özelliğin zorunlu olarak kod gerçekleştirimini yapması sağlanmıştır.

**public** **abstract** **int** YetkiDuzeyi { **get**; }

// Soyut üyelerde aşağıda görüldüğü gibi kod gerçekleştirimi bulunmamaktadır.

**public** **abstract** **bool**

YetkiSahibiMi(**string** işlem);

}

VB.NET

**MustInherit** **Class** YetkiliKullanıcı

**Inherits** TemelKullanıcı

' Salt okunur ve soyut olarak olarak tanımlanan bu özellik

' ile YetkiliKullanıcı sınıfını miras alan her sınıfın bu

' özelliğin zorunlu olarak kod gerçekleştirimini yapması sağlanmıştır.

**Public** **MustOverride** **ReadOnly** **Property** YetkiDuzeyi() **As** **Integer**

' Soyut üyelerde aşağıda görüldüğü gibi kod gerçekleştirimi bulunmamaktadır.

**Public** **MustOverride** **Function** YetkiSahibiMi(**ByVal** işlem **As** **String**) **As** **Boolean**

**End** **Class**

Kod 96, Soyut Özellik ve Metotlar

Üyelerin soyut olarak tanımlanması abstract ve MustOverride (sırasıyla C# ve VB.NET) rezerve kelimeleri ile yapılmaktadır.

Yönetici Kullanıcı sınıfı gerçekleştirimi aşağıdaki örnekte olduğu gibi yapılabilir.

C#

**class** YöneticiKullanıcı : YetkiliKullanıcı {

// Yetkilerin tutulduğu dizidir. Yetkilerin statik olarak sınıf bazında   
 // tanımlandığına dikkat ediniz.

**private** **static** ArrayList yetkiListesi;

// Statik yapıcı içerisinde yetki listesi oluşturmakta ve tüm nesneler   
 // için kullanıma hazırlanmaktadır.

**static** YöneticiKullanıcı() {

yetkiListesi = **new** ArrayList();

yetkiListesi.Add("Seçmen Ekleme");

yetkiListesi.Add("Seçmen Silme");

}

// override kelimesi ile ata sınıf tarafından soyut olarak tanımlanan   
 // özelliğin gerçeklekleştirimi yapılmaktadır.

**public** **override** **int** YetkiDuzeyi {

**get** { **return** 5; }

}

// Ata sınıf tarafından soyut olarak tanımlanan metot gerçekleştirimi   
 // yapılmıştır.

**public** **override** **bool** YetkiSahibiMi(**string** işlem) {

**return** yetkiListesi.IndexOf(işlem) >= 0;

}

}

// Kullanım örnekleri

YöneticiKullanıcı k = **new** YöneticiKullanıcı();

// Nesne tanımının YetkiliKullanıcı olarak yapılmasına rağmen oluşturulan   
// nesnenin YöneticiKullanıcı olduğuna dikkat ediniz.

YetkiliKullanıcı y = **new** YöneticiKullanıcı();

Console.WriteLine(k.YetkiDuzeyi);

Console.WriteLine(y.YetkiDuzeyi);

VB.NET

**Class** YöneticiKullanıcı

**Inherits** YetkiliKullanıcı

' Yetkilerin tutulduğu dizidir. Yetkilerin statik olarak sınıf bazında   
 ' tanımlandığına dikkat ediniz.

**Private** **Shared** yetkiListesi **As** ArrayList

' Statik yapıcı içerisinde yetki listesi oluşturmakta ve tüm nesneler için   
 ' kullanıma hazırlanmaktadır.

**Shared** **Sub** **New**()

yetkiListesi = **New** ArrayList()

**With** yetkiListesi

.Add("Seçmen Ekleme")

.Add("Seçmen Silme")

**End** **With**

**End** **Sub**

' override kelimesi ile ata sınıf tarafından soyut olarak tanımlanan   
 ' özelliğin gerçeklekleştirimi yapılmaktadır.

**Public** **Overrides** **ReadOnly** **Property** YetkiDuzeyi() **As** **Integer**

**Get**

**Return** 5

**End** **Get**

**End** **Property**

' Ata sınıf tarafından soyut olarak tanımlanan metot gerçekleştirimi   
' yapılmıştır.

**Public** **Overrides** **Function** YetkiSahibiMi(**ByVal** işlem **As** **String**) **As** **Boolean**

**Return** yetkiListesi.IndexOf(işlem) >= 0

**End** **Function**

**End** **Class**

' Kullanım örnekleri

**Dim** k **As** YöneticiKullanıcı = **new** YöneticiKullanıcı()

' Nesne tanımının YetkiliKullanıcı olarak yapılmasına rağmen oluşturulan   
' nesnenin YöneticiKullanıcı olduğuna dikkat ediniz.

**Dim** y **As** YetkiliKullanıcı = **new** YöneticiKullanıcı

Console.WriteLine(k.YetkiDuzeyi)

Console.WriteLine(y.YetkiDuzeyi)

Kod 97, Örnek Soyut Üye Gerçekleştirimi

Detayları polimorfizm konusunda aktarılmakla birlikte, kullanım örneği incelendiğinde, nesne tanımlamasının YetkiliKullanıcı olarak yapılmasına rağmen oluşturulan nesnenin YöneticiKullanıcı olduğu görülecektir.

Listelenen kaynak kodlar \Bölüm5\Abstracts klasöründe bulunmaktadır.

Kod çıktısı aşağıda gösterilmiştir.

5  
5

Dikkat edilmesi gereken önemli bir konu, derleyicinin y.YetkiDüzeyi ifadesi ile, y isimli değişken YetkiliKullanıcı tipinde olmasına rağmen, YöneticiKullanıcı.YetkiDüzeyi özelliğinin değerini döndürmesidir. Hangi sınıfın özelliğinin çağrılacağına çalışma zamanında karar verildiğine dikkat ediniz.

Yapıcı Metotlar ve Nesne Oluşturma

Ata sınıf tarafından belirlenen yapıcı metot da diğer üyeler gibi ata sınıfı miras alan sınıflar tarafından miras alınmaktadır.

Aşağıda yapıcı metot örneği ve nesne oluşturma sırasını anlatan örnek gösterilmiştir.

C#

**abstract** **class** TemelKullanıcı {

**private** **string** ad;

// Her kullanıcının bir adı olması gerektiği varsayımı ile temel

// kullanıcı sınıfına Ad isminde bir özellik eklenmiş ve

// yapıcıya bu değerin parametre olarak iletimi sağlanmıştır.

**protected** TemelKullanıcı(**string** ad) {

**this**.ad = ad;

Console.WriteLine("TemelKullanıcı oluşturuldu");

}

**public** **string** Ad {

**get** {

**return** ad;

}

// protected rezerve kelimesi ile hedeflenen, özelliğe

// değer atama hakkını sadece bu sınıfı miras alan sınıflara vermektir.

**protected** **set** {

ad = value;

}

}

}

**abstract** **class** YetkiliKullanıcı: TemelKullanıcı {

// base rezerve kelimesi ata sınıfa ulaşımı sağlamaktadır.

// Böylece TemelKullanıcı sınıfı yapıcı metodu çağrılmaktadır.

**protected** YetkiliKullanıcı(**string** ad)

: **base**(ad) {

Console.WriteLine("YetkiliKullanıcı oluşturuldu");

}

}

**class** YöneticiKullanıcı: YetkiliKullanıcı {

**public** YöneticiKullanıcı(**string** ad)

: **base**(ad) {

Console.WriteLine("YöneticiKullanıcı oluşturuldu");

}

}

// Kullanım Örneği

YöneticiKullanıcı k = **new** YöneticiKullanıcı("Mustafa");

Console.WriteLine("Kullanıcı Adı: {0}", k.Ad);

VB.NET

**MustInherit** **Class** TemelKullanıcı

**Private** \_ad **As** **String**

' Her kullanıcının bir adı olması gerektiği varsayımı ile temel

' kullanıcı sınıfına Ad isminde bir özellik eklenmiş ve yapıcıya bu değerin   
 ' parametre olarak iletimi sağlanmıştır.

**Protected** **Sub** **New**(**ByVal** ad **As** **String**)

**Me**.\_ad = ad

Console.WriteLine("TemelKullanıcı oluşturuldu")

**End** **Sub**

**Public** **Property** Ad() **As** **String**

**Get**

**Return** \_ad

**End** **Get**

' Protected rezerve kelimesi ile hedeflenen, özelliğe değer atama hakkını   
 ' sadece bu sınıfı miras alan sınıflara vermektir.

**Protected** **Set**(**ByVal** value **As** **String**)

\_ad = value

**End** **Set**

**End** **Property**

**End** **Class**

**MustInherit** **Class** YetkiliKullanıcı

**Inherits** TemelKullanıcı

' MyBase rezerve kelimesi ata sınıfa ulaşımı sağlamaktadır. Böylece   
 ' TemelKullanıcı sınıfı yapıcı metodu çağrılmaktadır.

**Protected** **Sub** **New**(**ByVal** ad **As** **String**)

**MyBase**.**New**(ad)

Console.WriteLine("YetkiliKullanıcı oluşturuldu")

**End** **Sub**

**End** **Class**

**Class** YöneticiKullanıcı

**Inherits** YetkiliKullanıcı

**Public** **Sub** **New**(**ByVal** ad **As** **String**)

**MyBase**.**New**(ad)

Console.WriteLine("YöneticiKullanıcı oluşturuldu")

**End** **Sub**

**End** **Class**

' Kullanım Örneği

**Dim** k **As** YöneticiKullanıcı

k = **New** YöneticiKullanıcı("Mustafa")

Console.WriteLine("Kullanıcı Adı: {0}", k.Ad)

Kod 98, Miraslık, Yapıcı Metot ve Nesne Oluşturma

Kod çıktısı aşağıda gösterilmiştir.

TemelKullanıcı oluşturuldu

YetkiliKullanıcı oluşturuldu

YöneticiKullanıcı oluşturuldu

Kullanıcı Adı: Mustafa

Bilindiği gibi sınıf içerisinden nesne referansına ulaşımda this ve Me (sırasıyla C# ve VB.NET) kelimeleri kullanılmaktadır. Benzer şekilde, bir sınıf içerisinde ata sınıfa ulaşımda da base ve MyBase (sırasıyla C# ve VB.NET) kelimeleri kullanılmaktadır.

Yukarıdaki örnekte TemelKullanıcı ve YetkiliKullanıcı sınıflarının yapıcı metotlarının protected olarak tanımlandığına dikkat ediniz. Bir zorunluluk olmamasına rağmen, bu metotların yapıcılarının dışarıya açık olmasını gerektirecek bir durum bulunmamaktadır.

Yapıcı metotların ata sınıf tarafından tanımlanmış olması, miras alan sınıf tarafından bu yapıcıların kullanılabilmesini sağlamakta ancak diğer üyeler gibi dışarıdan erişim imkanı vermemektedir.

| C# | VB.NET |
| --- | --- |
| **class** A {  **public** A(**int** x) {  }  **public** A() {  }  }  **class** B : A {   // Yapıcıya karakter dizi olarak  // gelen değer işlenerek tamsayıya  // çevrilmekte ve ata sınıf olan   // A sınıfı yapıcısına   // iletilmektedir.  **public** B(**string** s) :   **base**(**int**.Parse(s)) {  }  }  **class** C {  }  // Geçersiz Kullanımlar  C c = **new** C("5");  C c = **new** C(12);  B b = **new** B();  // Geçerli Kullanımlar  A a = **new** A(12);  B b = **new** B("5");  C c = **new** C(); | **Class** A  **Public** **Sub** **New**(**ByVal** x **As** **Integer**)  **End** **Sub**  **Public** **Sub** **New**()  **End** **Sub**  **End** **Class**  **Class** B  **Inherits** A  ' Yapıcıya karakter dizi olarak  ' gelen değer işlenerek tamsayıya  ' çevrilmekte ve ata sınıf olan   ' A sınıfı yapıcısına   ' iletilmektedir.  **Public** **Sub** **New**(**ByVal** s **As** **String**)  **MyBase**.**New**(**Integer**.Parse(s))  **End** **Sub**  **End** **Class**  **Class** C  **End** **Class**  ' Geçersiz Kullanımlar  **Dim** c **As** C = **New** C("5")  **Dim** c **As** C = **New** C(12)  **Dim** b **As** B = **New** B  ' Geçerli Kullanımlar  **Dim** a **As** A = **New** A(12)  **Dim** b **As** B = **New** B("5")  **Dim** c **As** C = **New** C |

Tablo 33, Miraslık ve Çeşitli Yapıcı Metot Örnekleri

A ve B sınıflarına ait yapıcıların C nesnesi oluşturmada kullanılamadığına dikkat edilmelidir.

Miras alan sınıf içerisinde, ata sınıf tarafından tanımlanan üyelere erişim direkt üye adı ile yapılabilir. Buna rağmen Base ve MyBase kelimeleri kullanarak derleyiciye özellikle ata sınıf kapsamında işlem yapması iletilebilir.

| C# | VB.NET |
| --- | --- |
| **class** A {  // Alan tanımlamaları  **protected** **int** i = 0;  **public** **string** s = "";  }  **class** B : A {  **private** **int** i = 12;  **void** M() {  // 12 değeri yazılacaktır.  Console.WriteLine(i);  Console.WriteLine(**this**.i);  // 0 değeri yazılacaktır.  Console.WriteLine(**base**.i);  // B sınıfında s isminde bir  // üye bulunmadığı için  // her iki kullanım da   // aynıdır.  Console.WriteLine(s);  Console.WriteLine(**base**.s);  }  } | **Class** A  ' Alan tanımlamaları  **Protected** i **As** **Integer** = 0  **Public** s **As** **String** = ""  **End** **Class**  **Class** B  **Inherits** A  **Private** i **As** **Integer** = 12  **Private** **Sub** M()  ' 12 değeri yazılacaktır.  Console.WriteLine(i)  Console.WriteLine(**Me**.i)  ' 0 değeri yazılacaktır.  Console.WriteLine(**MyBase**.i)  ' B sınıfında s isminde bir  ' üye bulunmadığı için  ' her iki kullanım da aynıdır.  Console.WriteLine(s)  Console.WriteLine(**MyBase**.s)  **End** **Sub**  **End** **Class** |

Tablo 34, Yapıcı Metot Giriş Niteleyici Örnekleri

Özellikle ata sınıf ve miras alan sınıfta aynı isme sahip üyelerin bulunması durumunda bu yöntem faydalıdır.

Miraslığın Engellenmesi

Sealed ve NotInheritable (sırasıyla C# ve VB.NET) ile bir sınıf tanımının donatılması, ilgili sınıfın miras alınabilmesini engellemektedir.

| C# | VB.NET |
| --- | --- |
| **sealed** **class** A {  }  // Aşağıdaki sınıf tanımı  // geçersizdir, A isimli sınıf // miras alınamaz, sealed olarak // tanımlanmıştır.  **class** B : A {  } | **NotInheritable** **Class** A  **End** **Class**  ' Aşağıdaki sınıf tanımı  ' geçersizdir, A isimli sınıf ' miras alınamaz, NotInheritable  ' olarak tanımlanmıştır.  **Class** B  **Inherits** A  **End** **Class** |

Kod 99, Miraslığın Engellenmesi

Miraslığın engellenmesi kod güvenliğinin sağlanmasında önemlidir. Temel prensip olarak, miras alınması istenmeyen veya böyle bir ihtiyacın olmadığı sınıflar miras alınamaz olarak tanımlanmalıdır.

Üye Gölgeleme

Ata sınıf tarafından tanımlanan özellik veya metotların, miras alan sınıfta aynı isim ve parametre listesi ile tanımlanması durumunda ata sınıfa ait üye örtülür (ing. Member Shadowing).

| C# | VB.NET |
| --- | --- |
| **class** A {  **public** **void** M() {  Console.WriteLine("A Sınıfı");  }  }  **class** B : A {  **public** **void** M() {  Console.WriteLine("B Sınıfı");  }  }  // Örnek Kullanımlar  // B sınıfı tipinde nesne  // oluşturulmakta ve M isimli metot // çağrılmaktadır.  B b = **new** B();  b.M();  // b nesnesi as kelimesi ile  // A sınıfına dönüştürülmekte ve // M isimli metot çağrılmaktadır.  (b **as** A).M();  // Polimorfizm çerçevesinde a isimli  // nesne A tipinde tanımlanmasına // rağmen B nesnesi  // oluşturulmaktadır  A a = new B();  a.M();  // a isimli nesne B nesnesine  // dönüştürülüp M metodu  // çağrılmaktadır.  (a as B).M(); | **Class** A  **Public** **Sub** M()  Console.WriteLine("A Sınıfı")  **End** **Sub**  **End** **Class**  **Class** B  **Inherits** A  **Public** **Sub** M()  Console.WriteLine("B Sınıfı")  **End** **Sub**  **End** **Class**  ' Örnek Kullanımlar  ' B sınıfı tipinde nesne  ' oluşturulmakta ve M isimli metot ' çağrılmaktadır.  **Dim** b **As** B = **New** B  b.M()  ' b nesnesi A nesnesine dönüştürülmekte  ' ve M isimli metot çağrılmaktadır.  **CType**(b, A).M()  ' Polimorfizm çerçevesinde a isimli  ' nesne A tipinde tanımlanmasına ' rağmen B nesnesi  ' oluşturulmaktadır  **Dim** a **As** A = **New** B  a.M()  ' a isimli nesne B sınıfına  ' dönüştürülüp M metodu  ' çağrılmaktadır.  **CType**(a, B).M() |

Tablo 35, Üye Gölgeleme Örneği

Örnek çıktısı aşağıda gösterilmiştir.

B Sınıfı

A Sınıfı

A Sınıfı

B Sınıfı

Çıktılar değerlendirildiğinde, nesne tanımlama ve oluşturma özelliklerine göre hangi metodun çağrımının yapılacağına derleme anında karar verildiği görülmektedir.

Ata sınıfın sahip olduğu metot veya özelliğin miras alan sınıf tarafından aynı isimle tanımlanarak gölgelenmesi durumunda derleyici tarafından uyarı verilecektir. Derleyici tarafından uyarı üretilmesinin önüne geçmek ve gölgelemenin özellikle yapıldığını belirtmek için, miras alan sınıfta bulunan üye tanımları new ve Shadows (sırasıyla C# ve VB.NET) kelimeleri ile donatılmalıdır.

| C# | VB.NET |
| --- | --- |
| **class** B : A {  **public** **new** **void** M() {  Console.WriteLine("B Sınıfı");  }  } | **Class** B  **Inherits** A  **Public** **Shadows** **Sub** M()  Console.WriteLine("B Sınıfı")  **End** **Sub**  **End** **Class** |

Gölgeleme yapılması durumunda derleyici sadece uyarı üretmekte, kaynak kod başarıyla derlenmektedir. Buna rağmen olası kullanım hatalarını engellemek için gölgelenen üye tanımlarının new ve Shadows kelimeleri ile donatılması önerilmektedir.

Sanal Üyeler

Miraslık konusunun, kodun tekrar kullanılabilirliği ve genişletilebilirliği temel hedefi göz önüne alındığında, en önemli öğelerinden birisi olarak değerlendirilen sanal üyeler, kullanım şekliyle metot ve özelliklerin gölgelenmesine benzemektedir.   
  
Bir metot veya özelliğin sanal olarak tanımlaması virtual ve Overridable (sırasıyla C# ve VB.NET) ile yapılmaktadır.

| C# | VB.NET |
| --- | --- |
| **class** A {  **public** **virtual** **void** M() {  // Temel kod gerçekleştirimi  // bu sınıf içerisinde   // yapılabilir.  Console.WriteLine("A Sınıfı");  }  }  **class** B : A {  **public** **override** **void** M() {  // Bu metot içerisinde temel  // sınıf içerisinde yapılan  // gerçekleştirim çağrılabilir,  // örneğin base.M() şeklinde.  Console.WriteLine("B Sınıfı");  }  }  // Örnek Kullanımlar  // B sınıfı tipinde nesne  // oluşturulmakta ve M isimli metot // çağrılmaktadır.  B b = **new** B();  b.M();  // b nesnesi as kelimesi ile  // A sınıfına dönüştürülmekte ve // M isimli metot çağrılmaktadır.  (b as A).M();  // Polimorfizm çerçevesinde a isimli  // nesne A tipinde tanımlanmasına // rağmen B nesnesi  // oluşturulmaktadır  A a = **new** B();  a.M();  // a isimli nesne B nesnesine  // dönüştürülüp M metodu  // çağrılmaktadır.  (a as B).M(); | **Class** A  **Public** **Overridable** **Sub** M()  ' Temel kod gerçekleştirimi  ' bu sınıf içerisinde   ' yapılabilir.  Console.WriteLine("A Sınıfı")  **End** **Sub**  **End** **Class**  **Class** B  **Inherits** A  **Public** **Overrides** **Sub** M()  ' Bu metot içerisinde temel  ' sınıf içerisinde yapılan  ' gerçekleştirim çağrılabilir,  ' örneğin MyMase.M() şeklinde.  Console.WriteLine("B Sınıfı")  **End** **Sub**  **End** **Class**  ' Örnek Kullanımlar  ' B sınıfı tipinde nesne  ' oluşturulmakta ve M isimli metot ' çağrılmaktadır.  **Dim** b **As** B = **New** B  b.M()  ' b nesnesi A nesnesine dönüştürülmekte  ' ve M isimli metot çağrılmaktadır.  **CType**(b, A).M()  ' Polimorfizm çerçevesinde a isimli  ' nesne A tipinde tanımlanmasına ' rağmen B nesnesi  ' oluşturulmaktadır  **Dim** a **As** A = **New** B  a.M()  ' a isimli nesne B sınıfına  ' dönüştürülüp M metodu  ' çağrılmaktadır.  **CType**(a, B).M() |

Tablo 36, Sanal Üyeler ve Kullanım Örneği

Kod çıktısı aşağıda gösterilmiştir.

B Sınıfı

B Sınıfı  
B Sınıfı

B Sınıfı

Gölgelemeden farklı olarak çağrımda kullanılan metodun, nesne tanımı ne olursa olsun oluşturulan nesnenin B sınıfına ait olduğuna dikkat edilmelidir.

Sanal özellik ve metotların temel kullanım amacı, ata sınıf tarafından sağlanan kod gerçekleştirimini genişletmektir. Ata sınıf tarafından sağlanan bir işlevin miras alan sınıf tarafından örtülmesi override ve Overrides (sırasıyla C# ve VB.NET) kelimeleri ile yapılmaktadır.

Ata sınıf, gölgelemeden farklı olarak, özellik ve metot tanımında ilgili üyenin sanal olduğunu belirtmektedir. Sanal tanımlanan üyeler için, yine gölgelemeden farklı olarak, çalışma anında hangi üyenin çağrılacağı belirlenmektedir.

Genel tasarım prensibi olarak, miras alan sınıflar tarafından genişletilebileceği değerlendirilen özellik ve metotlar sanal olarak tanımlanmalıdır. Buna rağmen, özellik veya metodun sanal olarak tanımlanması durumunda, sanal üye tablosu (ing. Virtual Member Table) adı verilen bir tablonun otomatik olarak çalışma zamanında yönetileceği unutulmamalıdır. Sanal üye tablosu yönetiminin çalışma anı performansına az da olsa negatif bir etkisi bulunmaktadır.

Aşağıda daha detaylı bir örnek gösterilmiştir.

| C# | VB.NET |
| --- | --- |
| **abstract** **class** TemelŞekil {  **public** **virtual** **void** Çiz() {  // Temel çizim işlemleri  // Örneğin çizim alanının  // temizlenmesi, çerçevenin  // çizimi gibi tüm şekillerde  // ortak olan işlemlerin  // yapılması  }  }  **class** Dikdörtgen : TemelŞekil {  **public** **override** **void** Çiz() {  // Ata sınıf işlevinin  // çağrılması  **base**.Çiz();  // Dikdörtgen çizimi  }  }  **class** Üçgen : TemelŞekil {  **public** **override** **void** Çiz() {  // Ata sınıf işlevinin  // çağrılması  **base**.Çiz();  // Üçgen çizimi  }  }  Console.WriteLine("Seçimler");  Console.WriteLine("d: Dikdörtgen");  Console.WriteLine("u: Üçgen");  **char** c =   Convert.ToChar(Console.Read());  // TemelŞekil tipinde bir nesne // tanımı yapılmaktadır.  TemelŞekil şekil;  // Kullanıcının seçimine göre // Dikdörtgen veya Üçgen nesneleri // oluşturulmaktadır. Burada // dikkat edilmesi gereken konu // kullanımın şekil.Çiz() şeklinde // olmasına rağmen, şekil değişkeni // ile hangi nesne ifade ediliyorsa // o nesnenin Çiz metodunu  // çağrımının yapılacağı olmalıdır.  **switch** (c) {  **case** 'd': {  şekil =  **new** Dikdörtgen();  **break**;  }  **case** 'u': {  şekil = **new** Üçgen();  **break**;  }  **default**:  **return**;  }  // şekil değişkeni ile ifade // edilen nesne derleme zamanında  // belirli değildir ve çalışma // zamanında belirlenmektedir.  şekil.Çiz(); | **MustInherit** **Class** TemelŞekil  **Public** **Overridable** **Sub** Çiz()  ' Temel çizim işlemleri  ' Örneğin çizim alanının  ' temizlenmesi, çerçevenin  ' çizimi gibi tüm şekillerde  ' ortak olan işlemlerin  ' yapılması  **End** **Sub**  **End** **Class**  **Class** Dikdörtgen  **Inherits** TemelŞekil  **Public** **Overrides** **Sub** Çiz()  ' Ata sınıf işlevinin  ' çağrılması  **MyBase**.Çiz()  ' Dikdörtgen çizimi  **End** **Sub**  **End** **Class**  **Class** Üçgen  **Inherits** TemelŞekil  **Public** **Overrides** **Sub** Çiz()  ' Ata sınıf işlevinin  ' çağrılması  **MyBase**.Çiz()  ' Üçgen çizimi  **End** **Sub**  **End** **Class**  Console.WriteLine("Seçimler")  Console.WriteLine("d: Dikdörtgen")  Console.WriteLine("u: Üçgen")  **Dim** c **As** **Char** =   Convert.ToChar(Console.Read)  ' TemelŞekil tipinde bir nesne ' tanımı yapılmaktadır.  **Dim** şekil **As** TemelŞekil  ' Kullanıcının seçimine göre ' Dikdörtgen veya Üçgen nesneleri ' oluşturulmaktadır. Burada ' dikkat edilmesi gereken konu ' kullanımın şekil.Çiz() şeklinde ' olmasına rağmen, şekil değişkeni ' ile hangi nesne ifade ediliyorsa ' o nesnenin Çiz metodunu  ' çağrımının yapılacağı olmalıdır.  **Select** **Case** (c)  **Case** "d"c  şekil = **New** Dikdörtgen()  **Case** "u"c  şekil = **New** Üçgen()  **Case** **Else**  **Return**  **End** **Select**  ' şekil değişkeni ile ifade ' edilen nesne derleme zamanında  ' belirli değildir ve çalışma ' zamanında belirlenmektedir.  şekil.Çiz() |

Sanal özellik veya metot tanımının sealed kelimesi ile donatılması, ilgili üyenin daha fazla örtülemeyeceğini belirtmektedir.

**class** A {

**public** **virtual** **void** M() {

}

}

**class** B : A {

**public** **override** **sealed** **void** M() {

}

}

**class** C : B {

// Kullanım geçersizdir, B sınıfı, M metodunun daha fazla örtülmesini  
 // sealed kelimesi ile engellemektedir.

**public** **override** **void** M() {

}

}

Polimorfizm

Şu ana kadar verilen örnekler ve özellikle miraslık konusunda aktarılan bazı konularda sık kullanılan polimorfizm, farklı tiplere sahip nesnelerin aynı şekilde ele alınabilme özelliğini ifade etmektedir.

Tip Dönüşümü

Genel kural olarak, ata sınıf ve miras alan sınıfların tip dönüşüm işleminde, miras alan sınıf tipinde oluşturulan nesne her zaman ata sınıfa dönüştürülebilir.

Dönüşüm kelimesi yanlış anlamalara sebep olmamalıdır. Miraslık konusunda da aktarıldığı üzere, miras alan sınıf ata sınıfın genişletilmiş halidir. Miras alan sınıfa ait bir nesnede de, ata sınıf tipinde oluşturulmuş bir nesnenin genişletilmiş şekli olarak düşünülebilir.

Polimorfizm Özellikleri

Miraslık konusunun pratikteki temel kullanım şeklini sağlayan polimorfizm ile ilgili çeşitli örnekler aşağıda gösterilmiştir.

C#

**abstract** **class** A {

// M isimli metot sanal olarak tanımlanmıştır.

**public** **virtual** **void** M() {

Console.WriteLine("A.M metodu");

}

// x isimli özellik soyut olarak tanımlanmıştır.

**public** **abstract** **int** X { **get**; }

**public** **void** N() {

Console.WriteLine( "A sınıfı N metodu");

}

}

**class** B : A {

**public** **override** **void** M() {

Console.WriteLine("B sınıfı M metodu");

}

**public** **override** **int** X {

**get** {

Console.WriteLine( "B sınıfı X Özelliği");

**return** 12;

}

}

}

**class** C : B {

**public** **new** **void** N() {

Console.WriteLine( "C sınıfı N metodu");

}

**public** **void** P() {

Console.WriteLine("C sınıfı P metodu");

}

}

// Örnekler ve Açıklamalar

A x = **new** B();

A y = **new** C();

B k = **new** B();

B z = **new** C();

// x nesnesi M metodu çağrılmaktadır. X değişkeninin tanım tipi A sınıfıdır   
// ancak oluşturulan nesne B nesnesidir. Polimorfizm kuralları uyarınca

// M metodu sanal olarak tanımlandığı ve B sınıfı tarafından örtüldüğü için

// çağrılan metot gerçekleştirimi B metoduna ait olacaktır.

x.M();

// B sınıfı tipinde tanımlanan z değişkeni için ata sınıf olarak

// B sınıfını kabul eden C tipinde nesne oluşturulmuştur.

((B)z).M();

// z nesnesi B sınıf tipinde tanımlanmış ve C nesnesi oluşturulmasına rağmen   
// gölgeleme yapıldığı için A sınıfının N metodu çağrılmaktadır. Bu kararın   
// derleme anında verildiğine dikkat edilmelidir.

// Yukarıdaki sanal metot ile farkına dikkat edilmelidir.

((B)z).N();

z.N();

((C)z).P();

((C)y).P();

// Hatalı tip dönüşümü

((C)k).P();

VB.NET

**MustInherit** **Class** A

' M isimli metot sanal olarak tanımlanmıştır.

**Public** **Overridable** **Sub** M()

Console.WriteLine("A.M metodu")

**End** **Sub**

' x isimli özellik soyut olarak tanımlanmıştır.

**Public** **MustOverride** **ReadOnly** **Property** X() **As** **Integer**

**Public** **Sub** N()

Console.WriteLine("A sınıfı N metodu")

**End** **Sub**

**End** **Class**

**Class** B

**Inherits** A

**Public** **Overrides** **ReadOnly** **Property** X() **As** **Integer**

**Get**

Console.WriteLine("B sınıfı X özelliği")

**Return** 12

**End** **Get**

**End** **Property**

**Public** **Overrides** **Sub** M()

Console.WriteLine("B sınıfı M metodu")

**End** **Sub**

**End** **Class**

**Class** C

**Inherits** B

**Public** **Shadows** **Sub** N()

Console.WriteLine("C sınıfı N metodu")

**End** **Sub**

**Public** **Sub** P()

Console.WriteLine("C sınıfı P metodu")

**End** **Sub**

**End** **Class**

' Örnekler ve Açıklamalar

**Dim** x **As** A = **New** B

**Dim** y **As** A = **New** C

**Dim** k **As** B = **New** B

**Dim** z **As** B = **New** C

' x nesnesi M metodu çağrılmaktadır. X değişkeninin tanım tipi A sınıfıdır ancak

' oluşturulan nesne B nesnesidir. Polimorfizm kuralları uyarınca M metodu sanal   
' olarak tanımlandığı ve B sınıfı tarafından örtüldüğü için çağrılan metot   
' gerçekleştirimi B metoduna ait olacaktır.

x.M()

' B sınıfı tipinde tanımlanan z değişkeni için ata sınıf olarak B sınıfını kabul   
' eden C tipinde nesne oluşturulmuştur.

**CType**(z, B).M()

' z nesnesi B sınıf tipinde tanımlanmış ve C nesnesi oluşturulmasına rağmen   
' gölgeleme yapıldığı için A sınıfının N metodu çağrılmaktadır. Bu kararın   
' derleme anında verildiğine dikkat edilmelidir. Yukarıdaki sanal metot ile

' farkına dikkat edilmelidir.

**CType**(z, B).N()

z.N()

**CType**(z, C).P()

**CType**(y, C).P()

' Hatalı tip dönüşümü

((C)k).P()*;*

Kod 100, Polimorfizm Örnekleri

Kod çıktısı aşağıda gösterilmiştir.

B sınıfı M metodu

B sınıfı M metodu

A sınıfı N metodu

A sınıfı N metodu

C sınıfı P metodu

C sınıfı P metodu

Kitabın farklı bölümlerinde yeri geldikçe polimorfizm konusunda detaylı açıklamalar yapılacaktır.

Kısmi Sınıflar

.NET 2.0 sürümünden itibaren aynı sınıfa ait tanım ve kod gerçekleştiriminin aynı kaynak kod dosyası içerisinde (.cs, .vb) bulunma zorunluluğu ortadan kalkmıştır.

Böylece, farklı kaynak kod dosyaları içerisinde aynı sınıfa ait tanımlamalar ve farklı üyelerin gerçekleştirimi yapılabilmekte, derleyici farklı kaynak kod dosyalarını tek bir dosya gibi ele alabilmektedir.

Şekil 15, Kısmi Sınıflar

Kısmi sınıf tanımı partial rezerve kelimesi ile yapılmaktadır.

| C# | VB.NET |
| --- | --- |
| // partial kelimesi ile yapılan // kısmi sınıf tanımı ile aynı  // sınıfın gerek aynı kaynak kod  // dosyası içerisinde, gerek farklı // kaynak kodyası içerisinde tekrar // tanımlanabilmesi sağlanmıştır.  **partial** **class** Test {  **public** **void** A() {  }  }  // Test isimli sınıf tekrar  // tanımlanmış ve B isimli metot // eklenmiştir.  **partial** **class** Test {  **public** **void** B() {  }  }  // Tekrar tanımlanan Test isimli // sınıf, bu tanımı ile ArrayList // sınıfını ata sınıf olarak kabul // etmektedir.  **partial** **class** Test : ArrayList {  }  // Kullanım Örnekleri ve Açıklamalar  // Test nesnesi oluşturulmaktadır.  Test t = **new** Test();  // A metodu çağrılmaktadır.  t.A();  // B metodu çağrılmaktadır.  t.B();  // Test sınıfı ArrayList sınıfını // miras aldığı için ArrayList.Add // metodu çağrılabilmektedir.  t.Add(12); | ' partial kelimesi ile yapılan ' kısmi sınıf tanımı ile aynı  ' sınıfın gerek aynı kaynak kod  ' dosyası içerisinde, gerek farklı ' kaynak kodyası içerisinde tekrar ' tanımlanabilmesi sağlanmıştır.  **Partial** **Class** Test  **Public** **Sub** A()  **End** **Sub**  **End** **Class**  ' Test isimli sınıf tekrar  ' tanımlanmış ve B isimli metot ' eklenmiştir.  **Partial** **Class** Test  **Public** **Sub** B()  **End** **Sub**  **End** **Class**  ' Tekrar tanımlanan Test isimli ' sınıf, bu tanımı ile ArrayList ' sınıfını ata sınıf olarak kabul ' etmektedir.  **Partial** **Class** Test  **Inherits** ArrayList  **End** **Class**  ' Kullanım Örnekleri ve Açıklamalar  ' Test nesnesi oluşturulmaktadır.  **Dim** t **As** Test = **New** Test  ' A metodu çağrılmaktadır.  t.A()  ' B metodu çağrılmaktadır.  t.B()  ' Test sınıfı ArrayList sınıfını ' miras aldığı için ArrayList.Add ' metodu çağrılabilmektedir.  t.Add(12) |

Tablo 37, Kısmi Sınıf Örnekleri

Kısmi sınıf kullanımı ile elde edilen avantajlar aşağıda listelenmiştir.

* Farklı programcıların aynı sınıfı farklı kaynak kod dosyaları içerisinde geliştirebilmeleri mümkündür.
* IDE tarafından otomatik kaynak kod üretimi ile oluşturulan sınıf gerçekleştiriminin farklı bir dosyada saklanması, programcının bu sınıfı genişletmek amacıyla yapacağı kod gerçekleştiriminin farklı bir dosyada saklanması ile kolay yönetilebilirlik sağlanmaktadır.

Sağlanan ikinci avantaj büyük yönetim ve kullanım kolaylığını beraberinde getirmektedir. Örneğin Windows uygulamaları içerisinde VS.NET tarafından otomatik olarak oluşturulan form kaynak dosyası ve programcının aynı sınıf içerisinde kod geliştirdiği dosyalar aşağıda gösterilmiştir.

Ekran Çıktısı 25, VS.NET Kısmi Sınıf Örneği

Ekran çıktısından, VS.NET çözüm penceresi içerisinde, form için Form1.vb ve Form1.Designer.vb isimli iki adet kaynak kod dosyası oluşturulduğu görülmektedir. Her iki dosya içerisinde de Form1 isimli sınıf tanımı bulunmaktadır. VS.NET, görsel tasarım yapılırken Form1.Designer.vb dosyası içerisine otomatik olarak forma eklenen kontrollerin (örneğin buton, etiket vb.) tanımlarını ve üretilen kaynak kodları eklemektedir.

Form1.vb isimli dosya ise programcı tarafından yönetilmekte, programcının eklediği kodlar bu dosyada tutulmaktadır. Form1.Designer.vb içerisinde Form1 sınıf tanımı kısmi olarak yapıldığı için derleyici tarafından her iki kaynak kod dosya içerikleri birleştirilmekte ve tek bir dosya gibi derleme işlemine devam edilmektedir.

VS.NET çözüm penceresi içerisinde Show All Files komutu verildiğinde Form1.Designer.vb dosyası görülebilir hale gelmektedir.

Kısmi sınıflar ile ilgili farklı özellikler ve kullanım sınırları aşağıda listelenmiştir.

* Bir sınıfa ait istenilen sayıda kısmi tanım yapılabilir.
* Derlemeden sonra IL kod içerisinde sınıfların kısmi tanımlandığına dair bilgi tutulmamaktadır. Çalışma zamanında kısmi sınıfların normal sınıflardan hiçbir farkı bulunmamaktadır.
* Kısmi tanımlanan sınıfların aynı Assembly içerisinde bulunma zorunluluğu vardır.
* Miras alınan sınıfın kısmi sınıf tanımlamalarından sadece birinde yapılması yeterlidir.
* Sınıf üyeleri aynı isim ve parametre listesi ile farklı kısmi sınıf tanımları içerisinde bulunamaz.

Arabirimler

Arabirimler, varlıkların ortak nitelik ve davranışlarını kod içerisinde tanımlayabilmeyi sağlayan yapılardır.

Sınıf ile arabirimin önemli görülen farkları aşağıda listelenmiştir.

* Sınıfların temel amacı gerçek dünya varlıklarını belirli soyutlama dereceleri ile yazılıma aktarmaktır. Bir sınıf tanımı içerisinde çeşitli alanlar, gerçekleştirimler, metotlar bulunabilir. Arabirimler içerisinde sadece özellik ve metot tanımları bulunabilir.
* Arabirimlerin temel özelliği, sınıflardan farklı olarak, varlıklar üzerinde değil, varlıkların nitelikleri ve davranışları üzerinde etkin olmalarıdır. Karşılaştırılabilme, sayılabilme, atanabilme, listelenebilme arabirim örnekleri olarak verilebilir.
* Bir sınıf bir veya birden fazla arabirim gerçekleştirimi yapabilir. Örneğin bir liste sınıfı gerek sayılabilir gerekse karşılaştırılabilir olabilir. Bir sınıfın arabirimi içermesi 'Sınıfın arabirim gerçekleştirimini yapması' (ing. Implementation of interface) olarak ifade edilmektedir.

Aşağıda örnek arabirim tanımlaması ve ilgili arabirimin gerçekleştirimini yapan sınıf gösterilmiştir.

C#

// Bir başka nesne ile karşılaştırılabilme davranışını belirten arabirim   
// tanımı yapılmıştır. Karsilastir isimli arabirim metoduna iletilen nesne,

// arabirimi gerçekleştiren nesne ile aynı ise metodun 0 döndürmesi  
// beklenmektedir.

**public** **interface** IKarsilastirilabilir {

**int** Karsilastir(**object** dest);

}

// IKarsilastirilabilir isimli arabirimin gerçekleştirimini yapan Test isimi   
// sınıf aşağıda gösterilmiştir.

**class** Test: IKarsilastirilabilir {

// Arabirimin tanımlanmasını zorunlu kıldığı Karsilastir metodu.

**public** **int** Karsilastir(**object** dest) {

// Karsilastir metodu kod gerçekleştiriminin bulunacağı metot bloğu.

}

}

VB.NET

' Bir başka nesne ile karşılaştırılabilme davranışını belirten arabirim   
' tanımı yapılmıştır. Karsilastir isimli arabirim metoduna iletilen nesne,  
' arabirimi gerçekleştiren nesne ile aynı ise metodun 0 döndürmesi   
' beklenmektedir.

**Public** **Interface** IKarsilastirilabilir

**Function** Karsilastir(**ByVal** dest **As** **Object**) **As** **Integer**

**End** **Interface**

' IKarsilastirilabilir isimli arabirimin gerçekleştirimini yapan Test isimi   
' sınıf aşağıda gösterilmiştir.

**Class** Test

**Implements** IKarsilastirilabilir

' Arabirimin tanımlanmasını zorunlu kıldığı Karsilastir metodu.

**Public** **Function** Karsilastir(**ByVal** dest **As** **Object**) **As** **Integer** \_ **Implements** IKarsilastirilabilir.Karsilastir

' Karsilastir metodu kod gerçekleştiriminin bulunacağı metot bloğu.

**End** **Function**

**End** **Class**

Kod 101, Arabirim Tanımı ve Örnek Sınıf Gerçekleştirimi

Bir sınıf, birden fazla arabirim gerçekleştirimi yapabilir. Bu durumda arabirimler , (virgül) karakteri ile ayrılmalıdır.

VB.NET için arabirim özellik ve metot isimleri ile sınıf içerisinde tanımlanan özellik ve metot isimlerinin aynı olma zorunluluğu bulunmamaktadır. Yukarıdaki örnekte  
  
Public Function KarsilastirXXX(ByVal dest As Object) As Integer  
Implements IKarsilastirilabilir.Karsilastir  
  
şeklinde de tanımlama geçerlidir. Buna rağmen arabirim üyeleri ile aynı isimli sınıf üyeleri kullanımı önerilmektedir.

Örnek Senaryo

Arabirim kullanım amacı ve detaylarını daha iyi aktarabilmek için anlatım örnek bir senaryo üzerinden yapılacaktır.

ABC web sitesi, farklı kitap depolarında bulunan kitapları kullanıcıların konularına göre sorgulayabildiği ve kitap deposuna siparişlerini iletebildiği bir platform sağlamaktadır. Böylece kullanıcılar sadece ABC web sitesini ziyaret ederek, ABC web sitesinin tüm iş ortaklarında bulunan kitapları tek noktadan sorgulama ve karşılaştırmalı fiyat bilgisi alma imkanına kavuşmaktadır.  
  
ABC web sitesi, depolardan kitap sorgulama ve sipariş iletimi için standart bir arabirim belirlemiştir. Kitap depoları bu arabirim gerçekleştirimini yapmakta, kaynak kodu kendileri geliştirmekte ve ABC web sitesine sadece Assembly dosyasını ve arabirim gerçekleştirimini yapan sınıf bütünleşik adını iletmektedirler. ABC web sitesi iletilen Assembly dosyasını sunucularına yüklemektedir.  
  
Farklı kitap depolarının ilettiği Assembly dosyaları, her kitap deposunun kendi sistemlerine erişim için belirlediği farklı yöntemleri içermektedir.

Senaryo incelendiğinde;

* ABC web sitesi, iş ortaklarına sorgu ve sipariş göndermek amacıyla, tüm iş ortakları için aynı olan bir arabirim tanımlamıştır.
* Farklı iş ortaklarına, kendi sistemlerine erişimi en uygun şekilde yapabilme imkanı tanınmıştır.
* ABC web sitesi, iş ortaklarının kendi sistemlerine erişimde izleyecekleri yöntemi belirlememiş, sadece kendisi ile iletişimde standart yöntem (arabirim ile) kullanma zorunluluğu getirmiştir.

Arabirim Tanımları

Şekil 16, Arabirim Örnek Senaryosu

Belirtilen ihtiyaç çerçevesinde ABC web sitesi tarafından tanımlanan arabirim aşağıda gösterilmiştir.

| C# | VB.NET |
| --- | --- |
| // Bir kitabın özelliklerini  // belirtir yapı tanımlanmıştır. // Her kitap için Isbn numarasının // tekil olduğu  // değerlendirilmektedir.  **public** **struct** Kitap {  **public** **string** Isbn;  **public** **string** Ad;  **public** **string** Yazar;  **public** **string** Konu;  **public** **float** Fiyat;  }  // ABC web sitesi tarafından  // herhangi bir kitap deposu // ile haberleşebilmek için // belirlenmiş arabirim tanımıdır.  **public** **interface** IKitapDeposu {  // Kitap deposu adını belirten   // salt okunur arabirim   // özelliğidir.  **string** DepoAd {  **get**;  }  // ABC web sitesinin ilgili  // kitap deposundan arama   // yapabilmesini sağlayan   // metottur.  Kitap[] AramaYap(**string** konu) {  }  // Kitap deposuna sipariş   // vermek için kullanılacak  // metottur.  **void** SiparisGonder(**string** isbn);  } | ' Bir kitabın özelliklerini  ' belirtir yapı tanımlanmıştır. ' Her kitap için Isbn numarasının ' tekil olduğu  ' değerlendirilmektedir.  **Public** **Structure** Kitap  **Public** Isbn **As** **String**  **Public** Ad **As** **String**  **Public** Yazar **As** **String**  **Public** Konu **As** **String**  **Public** Fiyat **As** **Single**  **End** **Structure**  ' ABC web sitesi tarafından  ' herhangi bir kitap deposu ' ile haberleşebilmek için ' belirlenmiş arabirim tanımıdır.  **Public** **Interface** IKitapDeposu  ' Kitap deposu adını belirten   ' salt okunur arabirim   ' özelliğidir.  **Property** **ReadOnly** DepoAd() \_  **As** **String**  ' ABC web sitesinin ilgili  ' kitap deposundan arama   ' yapabilmesini sağlayan   ' metottur.  **Function** AramaYap(**ByVal** konu \_  **As** **String**) **As** Kitap()  ' Kitap deposuna sipariş   ' vermek için kullanılacak  ' metottur.  **Sub** SiparisGonder(**ByVal** isbn \_  **As** **String**)  **End** **Interface** |

Tablo 38, Örnek Arabirim Senaryosu Tanımları

Kitap isimli yapı üyeleri kod sadeliğini sağlamak amacıyla dışarıya açık alanlar şeklinde açılmıştır. Kitabın daha önceki konularında belirtilen bilgi gizleme kuralları çerçevesinde, bu işlemin gizli alanlar – açık özellikler şeklinde yapılması önerilmektedir.

Kod Gerçekleştirimi

ABC web sitesinin çözüm ortaklarına iletilen Kitap tipi ve IKitapDeposu arabirimi, çözüm ortakları tarafından değerlendirilmiş ve bu arabirim gerçekleştirimini içeren farklı Assembly dosyaları ABC web sitesine gönderilmiştir.

| Çözüm Ortağı | Assembly | Sınıf Bütünleşik Adı |
| --- | --- | --- |
| A | ADeposu.dll | ADeposu.KitapEntegrasyon |
| B | Xyz.dll | Xyz.Sorgu |
| C | C.dll | Entegrasyonlar.Abc.KitapSorgu |

Tablo 39, Arabirim Senaryosu Çıktıları

Sonuç

ABC web sitesi, iletilen Assembly dosyalarını sunucularına yüklemiş ve çözüm ortakları bilgilerini sakladığı bir metin dosyasına yüklenen Assembly dosyalarının dizinlerini ve içerilerinde bulunan IKitapDeposu gerçekleştirimini içeren sınıf isimlerini tanımlamıştır.

# Örnek metin dosyası içeriği  
# Sınıf bütünleşik adı, Assembly Adı

ADeposu.KitapEntegrasyon,ADeposu

Xyz.Sorgu,Xyz

Entegrasyonlar.Abc.KitapSorgu,C

VS.NET içerisinde çözüm ve proje organizasyonu ile Assembly yapısı aşağıda gösterilmiştir.

Şekil 17, Arabirim Örnek Senaryo Assembly Yapısı

Assembly dosyası içerisinde bulunan sınıf tanımından çalışma zamanında nesne oluşturmak için Activator.CreateInstance metodu kullanılabilir.

C#

// IKitapDeposu nesnelerinin saklandığı liste tanımı

**private** ArrayList ortaklar = **new** ArrayList();

// Metin dosyayı okumak için StreamReader nesnesi kullanılmaktadır.

StreamReader r = **new** StreamReader(@"Ortaklar.txt");

// Metin dosyası son satırına kadar oku.

**while** (!r.EndOfStream) {

**string** line = r.ReadLine();

**if** (line.StartsWith("#"))

**continue**;

// Type.GetType metodu ile metin dosyada bulunan satır içerisindeki sınıf   
 // ve Assembly tanımından sınıf tipini al. Activator.CreateInstance  
 // metoduna bu tipi parametre geçerek sınıf tipinde bir nesne oluştur. Tip   
 // dönüşümü kurallarını kullanarak oluşturulan nesneyi IKitapDeposu   
 // arabirimine dönüştür.

IKitapDeposu depo = Activator.CreateInstance(Type.GetType(line))

**as** IKitapDeposu;

// Nesne başarıyla oluşturuldu ise listeye ekle.

**if** (depo != **null**)

ortaklar.Add(depo);

}

VB.NET

' IKitapDeposu nesnelerinin saklandığı liste tanımı

**Dim** ortaklar **As** ArrayList = **New** ArrayList()

' Metin dosyayı okumak için StreamReader nesnesi kullanılmaktadır.

**Dim** r **As** StreamReader = **New** StreamReader("Ortaklar.txt")

' Metin dosyası son satırına kadar oku.

**While** **Not** r.EndOfStream

**Dim** line **As** **String** = r.ReadLine()

**If** line.StartsWith("#") **Then**

**Continue** **While**

**End** **If**

' Type.GetType metodu ile metin dosyada bulunan satır içerisindeki sınıf   
 ' ve Assembly tanımından sınıf tipini al. Activator.CreateInstance  
 ' metoduna bu tipi parametre geçerek sınıf tipinde bir nesne oluştur. Tip   
 ' dönüşümü kurallarını kullanarak oluşturulan nesneyi IKitapDeposu   
 ' arabirimine dönüştür.

**Dim** depo **As** IKitapDeposu

depo **=** **CType**(Activator.CreateInstance(Type.**GetType**(line)), IKitapDeposu)

' Nesne başarıyla oluşturuldu ise listeye ekle.

**If** **Not** depo **Is** **Nothing** **Then**

ortaklar.Add(depo)

**End** **If**

**End** **While**

Kod 102, Activator.CreateInstance Kullanımı

Yukarıdaki kod örneği ile metin dosyada bulunan satırlar okunmuş ve her bir satırdaki tanıma göre arabirim gerçekleştirimi elde edilmiştir.

Çalışma anında farklı Assembly dosyaları içeriklerinin ve yapılarının okunması, herhangi bir nesneye ait üye bilgilerinin alınması gibi işlemler yansıma (ing. Reflection) olarak ifade edilebilir. Yukarıdaki örnekte de yansıma rutinleri kullanılarak Assembly içerisinde tanımlanan sınıf tipinde yeni bir nesne oluşturulmuştur.

Benzer kullanım örneği aşağıda gösterilmiştir.

C#

**string** tip = "ADeposu.KitapEntegrasyon,ADeposu";

IKitapDeposu depo = Activator.CreateInstance(Type.GetType(tip))   
 **as** IKitapDeposu;

// Kullanım:

Kitap [] sonuc;

sonuc = depo.AramaYap("zzz");

VB.NET

**Dim** tip **As** **String** = "ADeposu.KitapEntegrasyon,ADeposu"

**Dim** depo **As** IKitapDeposu

depo= **CType**(Activator.CreateInstance(Type.**GetType**(tip)) ,IKitapDeposu)

' Kullanım:

**Dim** sonuc() **As** Kitap

sonuc = depo.AramaYap("zzz")*;*

Listelenen kaynak kodlar \Bölüm5\Interfaces klasöründe bulunmaktadır.

Yukarıdaki kullanım detaylarında, her üç kitap deposunun da kendi standartlarında sınıf ve ad alanı kullandığını, farklı Assembly dosyaları ürettiklerini, buna rağmen ABC web sitesinin bu detaylarla hiçbir şekilde ilgilenmediğine dikkat edilmelidir. ABC web sitesi için iletilen Assembly içerisinden IKitapDeposu arabirimine dönüştürülebilen bir nesne oluşturmak yeterlidir.  
  
 Plugin olarak ifade edilen ve bir uygulama yazılımına sonradan yüklenerek yazılımın işlevselliğini artıran (örneğin Winamp Plugin gibi) yazılımların genel standardı arabirimler yardımıyla sağlanabilir mi ?

Arabirim Gerçekleştirimleri

Örnek senaryoda ana hatları ve somut kullanım örnekleri ile aktarılan arabirimler ve arabirim gerçekleştirimlerinin en sık kullanılan alanlarından birisi tip dönüşümü ve kontrolü kurallarıdır.

C#

// Konsola yazdırılabilmeyi niteleyen örnek arabirim

**interface** IYazdirilabilir {

**void** Yazdir();

}

**class** Test : IYazdirilabilir {

// Arabirim gerçekleştirimi

**public** **void** Yazdir() {

Console.WriteLine(**this**.GetType().Name);

}

}

**static** **class** Program {

**static** **void** Bilgi(**object** o) {

// Metoda iletilen nesnenin IYazdirilabilir arabirimini  
 // gerçekleştirdiğinin kontrolü

**if** (o **is** IYazdirilabilir) {

// Tip dönüşüm işlemi

IYazdirilabilir nesne = (IYazdirilabilir)o;

nesne.Yazdir();

}

**else**

Console.WriteLine(o.ToString());

}

**static** **void** Main(**string**[] args) {

**object** o1 = **new** SortedList();

**object** o2 = **new** Test();

Bilgi(o1);

Bilgi(o2);

}

}

VB.NET

' Konsola yazdırılabilmeyi niteleyen örnek arabirim

**Interface** IYazdirilabilir

**Sub** Yazdir()

**End** **Interface**

**Class** Test

**Implements** IYazdirilabilir

' Arabirim gerçekleştirimi

**Public** **Sub** Yazdir() **Implements** IYazdirilabilir.Yazdir

Console.WriteLine(**Me**.**GetType**().Name)

**End** **Sub**

**End** **Class**

**Class** Program

**Private** **Shared** **Sub** Bilgi(**ByVal** o **As** **Object**)

' Metoda iletilen nesnenin IYazdirilabilir arabirimini

' gerçekleştirdiğinin kontrolü

**If** (**TypeOf** (o) **Is** IYazdirilabilir) **Then**

**Dim** nesne **As** IYazdirilabilir = **CType**(o, IYazdirilabilir)

nesne.Yazdir()

**Else**

Console.WriteLine(o.ToString())

**End** **If**

**End** **Sub**

**Shared** **Sub** Main(**ByVal** args() **As** **String**)

**Dim** o1 **As** **Object** = **New** SortedList()

**Dim** o2 **As** **Object** = **New** Test()

Bilgi(o1)

Bilgi(o2)

**End** **Sub**

**End** **Class**

Kod 103, Arabirim Gerçekleştirim Örneği

Arabirim Miraslığı

Arabirimler, sınıflara benzer şekilde, soyutlama disiplini çerçevesinde miraslığı desteklemektedir.

| C# | VB.NET |
| --- | --- |
| **interface** ITest :   ICloneable, IComparable {  **void** TestMetot();  } | **Interface** ITest  **Inherits** ICloneable  **Inherits** IComparable  **Sub** TestMetot()  **End** **Interface** |

Kod 104, Arabirim Miraslığı

Arabirim miraslığının, sınıflardan farklı olarak, birden fazla arabirimi ata sınıf olarak kabul edebilmeyi desteklediğine dikkat ediniz.

Jenerikler

Jenerik (ing. Generic) tanımı aşağıda yapılmıştır.

Sınıflar, yapılar, arabirimler ve metot parametrelerinde değerler üzerinde farklı tiplerle çalışabilmeyi mümkün kılan yapıları jenerik olarak tanımlamak mümkündür.

C++ tanımı ile jenerikler, şablon (ing. Template) olarak ifade edilmektedir. .NET kapsamında bu tanım Generic olarak yapılmıştır. C++ şablonlarının işleyiş modeli ile .NET jeneriklerinin işleyiş modeli arasında önemli farklar bulunduğu unutulmamalıdır. C++ tarafından derleme zamanında yönetilen şablonlar, .NET içerisinde çalışma zamanında yönetilmektedir.

İhtiyaç Analizi

Aşağıda örnek liste sınıfları gösterilmiştir.

C#

// İçerisinde tamsayı tipleri saklayabilen örnek liste sınıfı

**class** IntegerListe {

// Listeye tamsayı bir değer eklemek için tanımlanan metot

**public** **void** Ekle(**int** eleman) {

}

// Liste içerisinde eleman isimli parametre ile belirtilen değeri

// arayan metot

**public** **int** Ara(**int** eleman) {

}

// Belirtilen pozisyonda bulunan tamsayı değerini döndüren

// özellik tanımı.

**public** **int** **this**[**int** index] {

**get** {

}

**set** {

}

}

}

// içerisinde karakter dizi tipleri saklayabilen örnek liste sınıfı

**class** StringListe {

// Listeye karakter dizi değer eklemek için tanımlanan metot

**public** **void** Ekle(**string** eleman) {

}

// Liste içerisinde eleman isimli parametre ile belirtilen değeri

// arayan metot

**public** **string** Ara(**string** eleman) {

}

// Belirtilen pozisyonda bulunan karakter dizi değerini döndüren

// özellik tanımı

**public** **string** **this**[**int** index] {

**get** {

}

**set** {

}

}

}

// Kullanım Örnekleri:

IntegerListe intList = **new** IntegerListe();

intList.Ekle(12);

intList.Ekle(30);

Console.WriteLine("{0}", intList[1]);

StringListe strList = **new** StringListe();

strList.Ekle("test1");

strList.Ekle("test2");

Console.WriteLine("{0}", strList[1]);

VB.NET

' İçerisinde tamsayı tipleri saklayabilen örnek liste sınıfı.

**Class** IntegerListe

' Listeye tamsayı bir değer eklemek için tanımlanan metot.

**Public** **Sub** Ekle(**ByVal** eleman **As** **Integer**)

**End** **Sub**

' Liste içerisinde eleman isimli parametre ile belirtilen değeri

' arayan metot.

**Public** **Function** Ara(**ByVal** eleman **As** **Integer**) **As** **Integer**

**End** **Function**

' Belirtilen pozisyonda bulunan tamsayı değerini döndüren

' özellik tanımı.

**Default** **Public** **Property** Item(**ByVal** index **As** **Integer**) **As** **String**

**Get**

**End** **Get**

**Set**(**ByVal** value **As** **String**)

**End** **Set**

**End** **Property**

**End** **Class**

' içerisinde karakter dizi tipleri saklayabilen örnek liste sınıfı.

**Class** StringListe

' Listeye karakter dizi değer eklemek için tanımlanan metot.

**Public** **Sub** Ekle(**ByVal** eleman **As** **String**)

**End** **Sub**

' Liste içerisinde eleman isimli parametre ile belirtilen değeri

' arayan metot

**Public** **Function** Ara(**ByVal** eleman **As** **String**) **As** **String**

**End** **Function**

' Belirtilen pozisyonda bulunan karakter dizi değerini döndüren

' özellik tanımı

**Default** **Public** **Property** Item(**ByVal** index **As** **Integer**) **As** **Integer**

**Get**

**End** **Get**

**Set**(**ByVal** value **As** **String**)

**End** **Set**

**End** **Property**

**End** **Class**

' Kullanım Örnekleri:

**Dim** intList **As** IntegerListe = **New** IntegerListe

intList.Ekle(12)

intList.Ekle(30)

Console.WriteLine("{0}", intList(1))

**Dim** strList **As** StringListe = **New** StringListe()

strList.Ekle("test1")

strList.Ekle("test2")

Console.WriteLine("{0}", strList(1))

Kod 105, Örnek Liste Sınıfları

Yukarıdaki örnek değerlendirildiğinde, içerisinde tamsayı ve karakter dizi değerleri saklayabilmek için iki ayrı liste sınıfı tanımlanmıştır. Her sınıfa ait, parametre tipleri farklılık gösteren Ekle ve Ara isimli metotlar tanımlanmıştır.

Sadece içerisinde saklanan tiplerin farklı olmasından dolayı farklı sınıf tanımları yapılmak zorunda kalınmıştır.

Benzer işlem, örneğin, ObjectListe isimli bir sınıf ve miraslık ile aşağıdaki şekilde yapılabilir.

C#

// Farklı tipte liste elemanlarını tüm nesnelerin atası olan object ile   
// yönetebilmek amacıyla tanımlanan sınıf

**class** ObjectListe {

// Ekle metodu, bu sınıfı miras alan sınıflar tarafındanörtülebilmesi için   
 // sanal olarak tanımlanmıştır.

**public** **virtual** **void** Ekle(**object** o) {

// Temel ekleme rutinlerinin bulunduğu blok

}

**public** **virtual** **object this**[**int** indeks] {

...

}

}

// Ata sınıf olarak ObjectListe sınıfını kabul eden tamsayı liste sınıfı

**class** IntegerListe: ObjectListe {

// ObjectListe.Ekle metodu bu sınıf tarafından örtülmüştür. Örtme

// yapabilmek için metot ismi ve parametre listesinin ata

// sınıftaki tanımı ile aynı olma zorunluluğunu hatırlayınız.

**public** **override** **void** Ekle(**object** o) {

// Listeye sadece tamsayılar eklenebilir. Parametre olarak iletilen   
 // nesne tamsayı ise ata sınıf metodu çağrılmakta ve ata sınıf

// gerekli işlemleri yapmaktadır.

**if** (o **is** **int**)

**base**.Ekle((**int**)o);

**else**

// İletilen parametre tamsayı değilse hata üret

**throw** **new** InvalidOperationException("Geçersiz tamsayı");

}

}

**class** StringListe : ObjectListe {

**public** **override** **void** Ekle(**object** o) {

. . .

}

}

// Kullanım Örnekleri:

// Nesne tanımlarının ObjectListe olarak yapıldığına dikkat ediniz.

ObjectListe intList = **new** IntegerListe();

intList.Ekle(12);

intList.Ekle(30);

// Listenin ilgili elemanını ObjectListe tarafından

// tanımlanan özellik yardımıyla al ve tip dönüşüm kuralları

// ile tamsayı tipe dönüştür.

**int** i = (**int**)intList[1];

ObjectListe strList = **new** StringListe();

strList.Ekle("test1");

strList.Ekle("test2");

**string** s = (**string**)strList[1];

VB.NET

' Farklı tipte liste elemanlarını tüm nesnelerin atası olan object ile   
' yönetebilmek amacıyla tanımlanan sınıf

**Class** ObjectListe

' Ekle metodu, bu sınıfı miras alan sınıflar tarafından örtülebilmesi için   
 ' sanal olarak tanımlanmıştır.

**Public** **Overridable** **Sub** Ekle(**ByVal** o **As** **Object**)

' Temel ekleme rutinlerinin bulunduğu blok

**End** **Sub**

**Default** **Public** **Overridable Property** Item(**ByVal** indeks **As** **Integer**) **As** **Object**

. . .

**End** **Property**

**End** **Class**

' Ata sınıf olarak ObjectListe sınıfını kabul eden tamsayı liste sınıfı

**Class** IntegerListe

**Inherits** ObjectListe

' ObjectListe.Ekle metodu bu sınıf tarafından örtülmüştür. Örtme yapabilmek   
 ' için metot ismi ve parametre listesinin ata sınıftaki tanımı ile aynı olma

' zorunluluğunu hatırlayınız

**Public** **Overrides**

**Sub** Ekle(**ByVal** o **As** **Object**)

' Listeye sadece tamsayılar eklenebilir. Parametre olarak iletilen nesne

' tamsayı ise ata sınıf metodu çağrılmakta ve ata sınıf gerekli   
 ' işlemleri yapmaktadır.

**If** **TypeOf** (o) **Is** **Integer** **Then**

**MyBase**.Ekle(**CType**(o, **Integer**)

)

**Else**

' İletilen parametre tamsayı değilse hata üret

**Throw** **New** InvalidOperationException("Geçersiz tamsayı")

**End** **If**

**End** **Sub**

**End** **Class**

**Class** StringListe

**Inherits** ObjectListe

**Public** **Overrides** **Sub** Ekle(**ByVal** o **As** **Object**)

. . .

**End** **Sub**

**End** **Class**

' Kullanım Örnekleri:

' Nesne tanımlarının ObjectListe olarak yapıldığına dikkat ediniz.

**Dim** intList **As** ObjectListe = **New** IntegerListe()

intList.Ekle(12)

intList.Ekle(30)

' Listenin ilgili elemanını ObjectListe tarafından tanımlanan özellik   
' yardımıyla al ve tip dönüşüm kuralları ile tamsayı tipe dönüştür.

**Dim** i **As** **Integer** = **CType**(intList(1), **Integer**)

**Dim** strList **As** ObjectListe = **New** StringListe()

strList.Ekle("test1")

strList.Ekle("test2")

**Dim** s **As** **String** = **CType**(strList(1), **String**)

Miraslığın kullanımı ile temel liste işlevlerinin ObjectListe sınıfı içerisinde yapılması sağlanmıştır. Buna rağmen, ObjectListe sınıfı her türlü nesneyi saklama yeteneğine sahip olduğu için, tip kontrollerinin IntegerListe ve StringListe içerisinde yapılma zorunluluğu ortaya çıkmıştır.

Yukarıdaki kullanım bir çok yönü ile ideal çözümü sağlasa da performans açısından değerlendirildiğinde bazı dezavantajları bulunmaktadır.

Kitabın daha önceki konularında bahsedilen kutulama ve kutudan çıkarma işlemi, listeye eklenen her eleman ve listeden alınan her eleman için yapılmak zorundadır.

Örneğin, IntegerListe.Ekle metodu object tipinde bir parametre değeri beklemektedir. .NET içerisindeki her yapının bir nesne olduğu kuralı ile bu metoda her tipte nesnenin iletimi mümkündür. Değer tiplerin (örneğin tamsayı) iletimi ile, değer tip kutulanmaktadır. Bir başka anlatımla, yeni bir hafıza ayrımı yapılarak nesne oluşturulmakta ve değer tip içeriği bu nesneye kopyalanmaktadır. Yeni oluşturulan nesne ObjectListe sınıfı tarafından (base.Ekle ve MyBase.Ekle metodu ile) saklanmakta, IntegerList.Ekle metoduna iletilen tamsayı değeri sadece kutulama amacıyla kullanılmaktadır.

Benzer şekilde, liste tarafından saklanan bir elemana erişildiğinde ve tamsayı tipe dönüştürüldüğünde ((int)intList[1] ve CType(intList(1), Integer)), arka planda yapılan işlem ObjectListe sınıfı tarafından saklanan nesneye erişim ve kutudan çıkartarak yeni bir tamsayı nesnesine ilgili içeriği kopyalamaktır.

Tanım ve Kullanım

Aşağıda jenerik sınıf örneği gösterilmiştir.

C#

// Jenerik Liste sınıf tanımı aşağıda yapılmıştır. T ile belirtilen sınıfın   
// saklayacağı elemanların tipidir. Sınıfın jenerik olarak tanımlanması ile   
// sınıf tipinde nesne tanımlarken tipin de belirtilme zorunluluğu   
// bulunmaktadır.

**class** Liste<T> {

// T ile belirtilen tipte olan eleman isimli parametre ve Ekle metodu

**public** **void** Ekle(T eleman) {

}

// T ile belirtilen tipte değer döndüren özellik tanımı.

**public** T **this**[**int** indeks] {

**get** {

. . .

}

**set** {

. . .

}

}

}

// Kullanım Örnekleri:

// intList adında, tamsayı değerleri saklayabilecek bir nesne oluştur.

Liste<**int**> intList = **new** Liste<**int**>();

// Ekle metodu, derleme zamanında da sadece tamsayı değerleri kabul   
// etmektedir.

intList.Ekle(12);

// Jenerik kullanımı ile tip dönüşümüne gerek kalmamıştır.

**int** i = intList[0];

// Aynı Liste sınıfı katakter dizi tipler için kullanılabilir.

Liste<**string**> strList = **new** Liste<**string**>();

strList.Ekle("asd");

**string** s = strList[0];

VB.NET

' Jenerik Liste sınıf tanımı aşağıda yapılmıştır. T ile belirtilen sınıfın   
' saklayacağı elemanların tipidir. Sınıfın jenerik olarak tanımlanması ile   
' sınıf tipinde nesne tanımlarken tipin de belirtilme zorunluluğu  
' bulunmaktadır.

**Class** Liste(**Of** T)

' T ile belirtilen tipte olan eleman isimli parametre ve Ekle metodu

**Public** **Sub** Ekle(**ByVal** eleman **As** T)

**End** **Sub**

' T ile belirtilen tipte değer döndüren özellik tanımı.

**Default** **Public**

**Property** Item(**ByVal** indeks **As** **Integer**) **As** T

**Get**

. . .

**End** **Get**

**Set**(**ByVal** value **As** T)

. . .

**End** **Set**

**End** **Property**

**End** **Class**

' Kullanım Örnekleri:

' intList adında, tamsayı değerleri saklayabilecek bir nesne oluştur.

**Dim** intList **As** Liste(**Of** **Integer**)

intList = **New** Liste(**Of** **Integer**)

' Ekle metodu, derleme zamanında da sadece tamsayı değerleri kabul

' etmektedir.

intList.Ekle(12)

'Jenerik kullanımı ile tip dönüşümüne gerek kalmamıştır.

**Dim** i **As** **Integer** = intList(0)

' Aynı Liste sınıfı katakter dizi tipler için kullanılabilir.

**Dim** strList **As** Liste(**Of** **String**)

strList = **New** Liste(**Of** **String**)

strList.Ekle("asd")

**Dim** s **As** **String** = strList(0)

Kod 106, Jenerik Tanım ve Kullanım Örnekleri

Bir sınıf, yapı veya arabirimi jenerik olarak tanımlamak için <> ve Of (sırasıyla C# ve VB.NET) ile tipi sınıf içerisinde belirten herhangi bir belirteç kullanılmaktadır.

Listelenen kodlardan da görüleceği üzere jenerik kullanımı çok daha sade, temiz ve daha kolay okunabilir bir kod yapısı sağlamaktadır.

Tip Zorlaması

Özellikle belirtilmediği durumlarda jenerikler her tipi kabul etmektedirler. Tip üzerinde jenerik sınıf tarafından işlem yapılması gerektiği durumlarda, jenerik sınıfa iletilen tipin özelliklerinin belirtilmesi gerekebilir.

| C# | VB.NET |
| --- | --- |
| Liste<**int**> intList= **new** Liste<**int**>();  Liste<**string**> strList =   **new** Liste<**string**>();  Liste<**double**> fList =   **new** Liste<**double**>(); | **Dim** intList **As** Liste(**Of** **Integer**)  intList = **New** Liste(**Of** **Integer**)  **Dim** i **As** **Integer** = intList(0)  **Dim** strList **As** Liste(**Of** **String**)  strList = **New** Liste(**Of** **String**)  **Dim** fList **As** Liste(**Of Double**)  fList = **New** Liste(**Of Double**) |

Bazı durumlarda jeneriklerin kabul edebilecekleri tipleri sınırlama ihtiyacı olabilir. Örneğin, bir jenerik liste sınıfının sadece IComparable arabirim gerçekleştirimini yapan sınıfları içerisinde saklaması istenebilir.

| C# | VB.NET |
| --- | --- |
| // Sadece IComparable arabirim // gerçekleştirimini yapan  // sınıfların geçerli olduğu // jenerik sınıf.  **class** Liste<T> where T: IComparable {  **public** **"** Ekle(T eleman) {  }  }  **class** Adres {  **string** Cadde;  }  // Sadece Adres ve Adres sınıfını // miras alan sınıfların geçerli  // olduğu jenerik sınıf.  **class** Test<T> where T : Adres {  } | ' Sadece IComparable arabirim ' gerçekleştirimini yapan  ' sınıfların geçerli olduğu ' jenerik sınıf.  **Class** Liste(**Of** T **As** IComparable)  **Public** **Sub** Ekle(**ByVal** eleman **As** T)  **End** **Sub**  **End** **Class**  **Class** Adres  **Private** Cadde **As** **String**  **End** **Class**  ' Sadece Adres ve Adres sınıfını ' miras alan sınıfların geçerli  ' olduğu jenerik sınıf.  **Class** Test(**Of** T **As** Adres)  **End** **Class** |

Tablo 40, Jenerikler Tip Zorlaması

Tip zorlaması where ve As (sırasıyla C# ve VB.NET) rezerve kelimeleri ile yapılmaktadır.

Jenerik tip zorlamasının temel kullanım amacı, arabirimler aracılığı ile nesnenin farklı özellik ve metotlarına erişim sağlamaktır.

| C# | VB.NET |
| --- | --- |
| **Class** Liste<T> **where** T: IComparable {  **public** **void** Ekle(T eleman) {  // eleman isimli parametrenin  // IComparable   // gerçekleştirimini yaptığı  // derleme zamanında   // belirlidir. Örneğin  // eleman.CompareTo şeklinde  // kullanım bu durumda   // geçerlidir.  }  } | **Class** Liste(**Of** T **As** IComparable)  **Public** **Sub** Ekle(**ByVal** eleman **As** T)  ' eleman isimli parametrenin  ' IComparable   ' gerçekleştirimini yaptığı  ' derleme zamanında   ' belirlidir. Örneğin  ' eleman.CompareTo şeklinde  ' kullanım bu durumda   ' geçerlidir.  **End** **Sub**  **End** **Class** |

Özet ve Öneriler

Bu bölümün tamamlanması ile NYP temelleri ve bu temellerin .NET içerisinde somut uygulama yöntemleri hakkında bilgiler verilmiştir.

NYP ve bu disiplinin .NET ile pratikte uygulanabilmesi önemlidir. Bir sonraki bölüme geçilmeden önce kısa bir tekrar ve kodlama çalışmaları yapılması önerilmektedir.

İlk olarak, açık bir şekilde bilmediğim bir şeyi asla doğru olarak kabul etmem. İkinci olarak, doğru çözüme ulaşmak için incelediğim konuyu mümkün olduğu ölçüde küçük parçalara ayırır, ondan sonra analiz ederim. Üçüncü olarak, küçükten başlayarak büyüğe doğru adım adım ilerleyerek düşüncelerimi netleştirmeye çalışırım. Nihayet, son olarak, her durumun sonucunu ortaya koyar ve genel olarak gözden geçiririm.

Rene Descartes

Bölüm

6

İleri Seviye Nesne İşlemleri

N

YP özelliklerinin ve .NET içerisinde C# ve VB.NET dilleri ile kullanım yöntemlerinin aktarıldığı bir önceki bölümün ardından, bu bölümde daha ileri seviye işlemler hakkında bilgiler verilecektir.

.NET programlama konusuna yeni başlayan okuyucuların bu bölüme geçmeden önce çeşitli kodlama çalışmaları yapmış olmaları beklenmektedir.

Hafıza Yönetimi ve Nesne Yaşam Süreci

Yönetilen bir platform sağlayan .NET, hafıza yönetimi ve nesnelerin yaşam sürecini belirli kurallar çerçevesinde kendisi düzenlemektedir.

CLR tarafından yapılan hafıza yönetiminde temel strateji, programcı tarafından oluşturulan nesnelerin gerekmediği durumlarda otomatik olarak belirli zaman aralıklarında GC (Garbage Collector) tarafından temizlenmesidir. Bir önceki konuda sunulan örneklerde nesneler new kelimesi ile oluşturulmasına rağmen, bir başka anlatımla new kelimesi ile hafıza ayrımı yapılmış olmasına rağmen, programcıdan oluşturduğu nesneleri yok etmesi beklenmemektedir.

Referans ve Değer Tipler

Bilindiği gibi referans tipler (System.ValueType sınıfını miras almayan tipler) heap üzerinde, değer tipler ve yerel değişkenler ise stack üzerinde saklanmaktadır.

Değer tipler ve yerel değişkenler için tanımlandıkları anda otomatik olarak stack üzerinde gerekli ayrım yapılmakta, metot çıkışında ise ayrılan stack üzerinden atılmaktadır.

Referans tipler iki fazda değerlendirilebilir.

* Referans tipte olan değişken tanımı (nesne tanımı)
* New kelimesi ile nesnenin oluşturulması

Yerel değişken tanımı olarak da ifade edilebilecek nesne tanımı hafıza yönetim kuralları uyarınca stack üzerinde saklanmaktadır. Buna rağmen new kelimesi ile nesne oluşturulduğunda, nesne için ayrılan hafıza heap üzerindedir.

C#

**class** Kitap {

**private** **string** ad, yazar;

**public** Kitap(**string** ad, **string** yazar) {

**this**.ad = ad;

**this**.yazar = yazar;

}

**public** **override** **string** ToString() {

**return** **string**.Format("Ad: {0}, Yazar: {1}", ad, yazar);

}

}

**struct** Nokta {

**public** **int** X;

**public** **int** Y;

}

**static** **class** Program {

**static** **void** Test() {

**int** i;

Kitap k;

Nokta n;

k = **new** Kitap("Köprü", "Ayşe Kulin");

}

}

VB.NET

**Class** Kitap

**Private** \_ad **As** **String**

**Private** \_yazar **As** **String**

**Public** **Sub** **New**(**ByVal** ad **As** **String**, **ByVal** yazar **As** **String**)

**Me**.\_ad = ad

**Me**.\_yazar = yazar

**End** **Sub**

**Public** **Overrides** **Function** ToString() **As** **String**

**Return**

**String**.Format("Ad: {0}, Yazar: {1}", \_ad, \_yazar)

**End** **Function**

**End** **Class**

**Structure** Nokta

**Public** X **As** **Integer**

**Public** Y **As** **Integer**

**End** **Structure**

**Class** Program

**Shared** **Sub** Test()

**Dim** i **As** **Integer**

**Dim** k **As** Kitap

**Dim** n **As** Nokta

k = **New** Kitap("Köprü", "Ayşe Kulin")

**End** **Sub**

**End** **Class**

Şekil 18, Stack ve Heap

Yukarıdaki örnekte Kitap tipindeki k isimli değişkenin stack üzerinde, bu değişkenin gösterdiği (işaret ettiği) nesnenin heap üzerinde saklandığına dikkat ediniz.

Test isimli metottan çıkışta, stack üzerine eklenen değerler otomatik olarak atılmakta, böylece değer tipler için hafıza temizlenmektedir. Buna rağmen kitap nesnesi heap üzerinde kalmaktadır.

Nesne Yaşam Süreci

Yukarıda verilen örnekte, k isimli değişkenin de stack üzerinde tutulduğu ve metot çıkışından sonra geçersiz olduğuna dikkat ediniz. Buna rağmen k isimli değişken geçersizleştiği an kitap nesnesi heap üzerinden temizlenmemiştir. Heap üzerinde tutulan nesneler, detayları bu kitabın konusu dışında olan ve olabildiğince optimize edilmiş algoritmalar ile "nesneye referansın kalmaması" ilkesi ile otomatik olarak GC tarafından temizlenmektedir. Böylece, programcının kendi oluşturduğu nesneyi yok etme (örneğin C++ delete kelimesi ile yapılan) işlemlerini yapması gerekmemektedir.

GC tarafından yapılan hafıza temizleme işlemi, belirli zaman aralıklarında veya hafızada bir nesneyi oluşturacak yer kalmadığında otomatik olarak yapılmaktadır. Bir nesnenin ne zaman hafızadan atılacağının herhangi bir ölçütü bulunmamaktadır. Nesneye referans kalmasa dahi, GC optimizasyon kuralları gereğince, belirli bir süre daha hafızada kalmaya devam edebilir, bir sonraki hafıza temizleme işleminde hafızadan atılmayabilir.  
  
Hafızadan atılma işleminin yapılabilmesi için gerekli koşul, oluşturulan nesneye uygulama içerisinden hiç bir referansın kalmamasıdır. Özellikle web tabanlı ve çok kullanıcılı uygulamalarda, hafızada büyük alan kaplayan nesnelere gereksiz referansların (örneğin statik olarak sınıf içerisinden) bulunması, bu nesnelerin hafızadan atılamamasını da beraberinde getirmektedir.

Sonlandırılabilme ve Yok Edilebilme

Yönetilen ortamın sağladığı otomatik hafıza temizleme işlemi sadece yönetilen nesneler için geçerlidir. Bir başka anlatımla, sadece CLR tarafından yapılan hafıza ayırma işlemi otomatik olarak yine CLR tarafından temizlenebilir.

Bazı işlemlerde, yönetilmeyen ortam çağrıları yapmak gerekebilir. Örneğin, .NET içerisinden bir dosyaya erişmek gerektiğinde, işletim sistemi tarafından sağlanan API fonksiyonlarının kullanımı zorunludur (Disk yönetimi işletim sistem tarafından yapılmaktadır). Yönetilmeyen ortam olan işletim sistemi doğal olarak otomatik hafıza yönetimini desteklememektedir. Bu gibi durumlarda, nesnenin yok edilmesi aşamasında yönetilmeyen ortamda ayrılmış kaynakları boşaltmak gerekebilir.

Object.Finalize metodu ve IDisposable arabirimi, nesnelerin hafızadan temizlenme aşamasında haberdar olmak ve gereken işlemleri yapmak için kullanılmalıdır.

Sonlandırılabilme

Object.Finalize metodunu özelleştiren nesneler sonlandırılabilir (ing. Finalizable) olarak ifade edilmektedir. Sonlandırılabilir nesnelerin tek kullanım amacı yönetilmeyen kaynakların (örneğin dosya açık ise kapatma işlemi gibi), nesnenin GC tarafından hafızadan temizlenmesi aşamasında, programcı tarafından temizlenmesinin sağlanmasıdır.

Geliştirilen uygulamada sadece .NET nesneleri kullanılıyorsa bu metodu özelleştirmenin gereği bulunmamaktadır.

Uygulama yazılımlarının neredeyse tamamı, alt seviye işlemler yapılmadığı sürece, sonlandırılabilir nesnelere ihtiyaç duymamaktadır. Bir nesnenin sonlandırılabilir olması durumunda, ilgili nesne CLR tarafından sonlandırılabilir olarak işaretlenmekte ve GC tarafından bu nesnenin hafızadan atılma süreci farklı işlemektedir. Performansa önemli ölçüde negatif etkisi olan sonlandırılabilir nesnelerin sadece gerektiğinde kullanımı önerilmektedir.

Object.Finalize metodu protected virtual olarak tanımlanmıştır ve içerisinde herhangi bir kod gerçekleştirimi bulunmamaktadır.

| C# | VB.NET |
| --- | --- |
| **class** Dosya {  **public** Dosya(**string** ad) {  }  **public** **void** Ac() {  // Bu metot içerisinde   // yönetilmeyen ortama   // çağrılarda bulunulabilir.   // Örneğin işletim sistemi  // tarafından sağlanan API   // çağrıları ile dosya   // açılabilir. Dosyanın açılması  // ile işletim sistemi   // tarafından gerekli hafıza  // ayrımı yapılmakta ve   // uygulama yazılımının açtığı  // dosyayı kapatması   // beklenecektir.  }  **public** **void** Kapat() {  // Programcı tarafından metodun  // çağrılması ile, açılan bir   // dosya metot yardımıyla   // kapatılabilir. Buna rağmen,  // programcının bu metodu  // çağırmayı unutması durumunda  // dosya açık kalacaktır. Bu   // durumda aşağıdaki metot  // bu işi yapmaktadır.  }  ~Dosya() {  // Yönetilmeyen kaynak temizleme  // işleminin yapıldığı metot.  // Bu metot içerisinde, örneğin,  // dosyanın hala açık olduğu  // kontrolü yapılabilir ve açık  // ise nesnenin Kapat metodu  // çağrılabilir.  }  } | **Class** Dosya  **Public** **Sub** **New**(**ByVal** ad **As** **String**)  **End** **Sub**  **Public** **Sub** Ac()  ' Bu metot içerisinde   ' yönetilmeyen ortama   ' çağrılarda bulunulabilir.   ' Örneğin işletim sistemi  ' tarafından sağlanan API   ' çağrıları ile dosya   ' açılabilir. Dosyanın açılması  ' ile işletim sistemi   ' tarafından gerekli hafıza  ' ayrımı yapılmakta ve   ' uygulama yazılımının açtığı  ' dosyayı kapatması   ' beklenecektir.  **End** **Sub**  **Public** **Sub** Kapat()  ' Programcı tarafından metodun  ' çağrılması ile, açılan bir   ' dosya metot yardımıyla   ' kapatılabilir. Buna rağmen,  ' programcının bu metodu  ' çağırmayı unutması durumunda  ' dosya açık kalacaktır. Bu   ' durumda aşağıdaki metot  ' bu işi yapmaktadır.  **End** **Sub**  **Protected** **Overrides** **Sub** Finalize()  ' Yönetilmeyen kaynak temizleme  ' işleminin yapıldığı metot.  ' Bu metot içerisinde, örneğin,  ' dosyanın hala açık olduğu  ' kontrolü yapılabilir ve açık  ' ise nesnenin Kapat metodu  ' çağrılabilir.  **End** **Sub**  **End** **Class** |

Tablo 41, Sonlandırılabilir Nesneler

C++ programlama konusunda deneyimli okuyucular için ~ karakteri yabancı değildir. Yok edici metodu (ing. Destructor) ifade edilen bu işlemin, otomatik hafıza yönetiminin bulunmadığı C++yok edici metotları işleyiş yöntemi ile karıştırılmaması önemlidir.

Yok Edilebilme

Nesnelerin hafızadan temizlenme aşamasında yönetilmeyen kaynakların temizlenmesi için bir diğer alternatif de IDisposable arabirimidir.

**public** **interface** IDisposable {

**void** Dispose();

}

IDisposable arabirimi, Dispose isminde bir metot içermektedir. Bu arabirim gerçekleştirimini yapan nesneler, yok edilebilir nesne (ing. Disposable Object) olarak ifade edilmektedir.

IDisposable arabiriminin temel amacı, nesne yok etme sürecinde programcı tarafından yapılmak istenen işlemleri tüm sınıflar için standart bir yapıya kavuşturmaktır.

C#

**void** YokEt(**object** o) {

**if** (o **is** IDisposable)

((IDisposable)o).Dispose();

}

VB.NET

**Sub** YokEt(**ByVal** o **As** **Object**)

**If** **TypeOf** (o) **Is** IDisposable **Then**

**CType**(o, IDisposable).Dispose()

**End** **If**

**End** **Sub**

Kod 107, System.IDisposable Kullanım Örneği

Yukarıdaki örnekte YokEt metoduna iletilen parametrenin IDisposable gerçekleştirimini yaptığı kontrol edilmekte, eğer nesne bu gerçekleştirimi yaptı ise Dispose metodu çağrılmaktadır.

IDisposable arabirim kullanımının istisna yönetim kodu ile yapılması tercih edilebilir.

| C# | VB.NET |
| --- | --- |
| // IDisposable arabirim  // gerçekleştirimini yap.  **class** Dosya: IDisposable {  . . .  **public** **void** Dispose() {  // Yönetilmeyen kaynak temizleme  // işlemi bu metot içerisinde  // yapılabilir. Bu metot   // içerisinde aynı zamanda Dosya  // sınıfının içerdiği diğer yok  // edilebilir nesnelerin Dispose  // metodu da çağrılabilir.  }  . . .  }  **static** **class** Program {  **static** **void** Main() {  Dosya d = **new** Dosya();  **try** {  d.Ac();  // Nesnenin farklı metotları  // çağrılabilir.  }  **finally** {  // Her durumda Dispose   // metodunun çağrılacağı   // garanti altındadır.  d.Dispose();  }  }  } | ' IDisposable arabirim  ' gerçekleştirimini yap.  **Class** Dosya  **Implements** IDisposable  **Public** **Sub** Dispose()   **Implements** IDisposable.Dispose  ' Yönetilmeyen kaynak temizleme  ' işlemi bu metot içerisinde  ' yapılabilir. Bu metot   ' içerisinde aynı zamanda Dosya  ' sınıfının içerdiği diğer yok  ' edilebilir nesnelerin Dispose  ' metodu da çağrılabilir.  **End** **Sub**  **End** **Class**  **Class** Program  **Shared** **Sub** Main()  **Dim** d **As** Dosya = **New** Dosya()  **Try**  d.Ac()  ' Nesnenin farklı metotları  ' çağrılabilir.  **Finally**  ' Her durumda Dispose   ' metodunun çağrılacağı   ' garanti altındadır.  d.Dispose()  **End** **Try**  **End** **Sub**  **End** **Class** |

Kod 108, İstisna Yönetimi ve IDisposable

Yukarıdaki örnekte dosya açma veya okuma gibi işlemlerde bir hata oluşması durumunda dahi, Dispose metodunun Finally bloğu içerisinde kullanımı ile metodun çağrılacağı güvence altına alınmıştır.

IDisposable arabirim gerçekleştirimini yapan bir nesnenin en doğru kullanımı Using bloğu içerisinde nesne tanım ve oluşturma işlemini yapmaktır. Using bloğu içerisinde oluşturulan bir nesnenin, IDisposable arabirim gerçekleştirimini yapması durumunda, blok çıkışında otomatik olarak Dispose metodu otomatik olarak çağrılmaktadır.

Yukarıdaki örnekteki Main metodu içeriğinin birebir karşılığı aşağıda gösterilmiştir.

| C# | VB.NET |
| --- | --- |
| **using** (Dosya d = **new** Dosya()) {  d.Ac();  }  // Blok çıkışında otomatik olarak  // Dispose metodu çağrılmaktadır. | **Using** d **As** Dosya = **New** Dosya()  d.Ac()  **End** **Using**  ' Blok çıkışında otomatik olarak  ' Dispose metodu çağrılmaktadır. |

Kod 109, Using Bloğu Kullanımı

Using bloğu, daha temiz ve okunabilir kaynak kod yapısı sağlama avantajının yanında, programcının yok edilebilir nesne kullandığında, Dispose metodunu çağırmayı unutması durumunu da ortadan kaldırmaktadır.

Ad alanlarının belirtilmesinde kullanılan C# using bloğu ile yukarıda aktarılan using bloğu birbirine karıştırılmamalıdır.

Sonlandırılabilir nesnelerde, hafızadan nesnenin atılması aşamasında CLR tarafından otomatik olarak Finalize metodunun çağrıldığı, yok edilebilir nesne kullanımında ise Using bloğu ile Dispose metodunun otomatik olarak çağrıldığı görülmektedir. Buna rağmen, yok edilebilir nesne kullanımında programcının using bloğunu tercih etmemesi ve Dispose metodunu da çağırmaması durumunda yönetilmeyen kaynakları temizleme işlemi yapılmayabilir.   
  
IDisposable arabirimi kullanımı ile, nesne tarafından oluşturulan ve bu arabirim gerçekleştirimini yapan diğer nesnelerin de Dispose metodu içerisinde yok edilebileceği unutulmamalıdır.

Birlikte Kullanım

Sonlandırılabilme ve yok edilebilme özelliklerinin birlikte kullanımı ile ilgili Microsoft tarafından belirlenen şablon ve gerekli açıklamalar aşağıda gösterilmiştir.

C#

**class** OrnekSablon : IDisposable {

// Nesnenin yok edilip edilmediğini belirten alan.

**private** **bool** disposed = **false**;

**public** **void** Dispose() {

// Temizle metodu true değeri ile çağrılmaktadır. Böylece, yok etme   
 // işleminin kullanıcı tarafından Dispose metodunu çağırarak   
 // (veya using bloğu tarafından) otomatik olarak yapıldığı belirtilmektedir.

Temizle(**true**);

// SuppressFinalize metodunun çağrılması ile GC tarafından bu nesne için   
 // sonlandırma işleminin yapılmaması gerektiği belirtilmektedir.

// Böylece, Finalize metodu çağrılmamakta, performans açısından negatif   
 // etki ortadan kalkmaktadır.

GC.SuppressFinalize(**this**);

}

**void** Temizle(**bool** disposing) {

// Temizleme işleminin daha önce yapılıp yapılmadığı kontrol edilmektedir.

**if** (!**this**.disposed) {

**if** (disposing) {

// Burada nesne tarafından oluşturulan nesnelerin Dispose metotları

// çağrılabilir.

}

// Yönetilmeyen kaynakların temizleneceği bölüm.

}

disposed = **true**;

}

~OrnekSablon() {

// Temizle metodu false değeri ile çağrılarak, GC tarafından bu çağrımın

// yapıldığı belirtilmektedir.

Temizle(**false**);

}

}

VB.NET

**Class** OrnekSablon

**Implements** IDisposable

' Nesnenin yok edilip edilmediğini belirten alan.

**Private** disposed **As** **Boolean** = **False**

**Public** **Sub** Dispose()

**Implements** IDisposable.Dispose

' Temizle metodu true değeri ile çağrılmaktadır. Böylece, yok etme   
 ' işleminin kullanıcı tarafından Dispose metodunu çağırarak   
 ' (veya using bloğu tarafından otomatik olarak) yapıldığı belirtilmektedir.

Temizle(**True**)

' SuppressFinalize metodunun çağrılması ile GC tarafından bu nesne için   
 ' sonlandırma işleminin yapılmaması gerektiği belirtilmektedir.

' Böylece, Finalize metodu çağrılmamakta, performans açısından negatif   
 ' etki ortadan kalkmaktadır.

GC.SuppressFinalize(**Me**)

**End** **Sub**

**Private** **Sub** Temizle(**ByVal** disposing **As** **Boolean**)

' Temizleme işleminin daha önce yapılıp yapılmadığı kontrol edilmektedir.

**If** **Not** **Me**.disposed **Then**

**If** disposing **Then**

' Burada nesne tarafından oluşturulan nesnelerin Dispose metotları

' çağrılabilir.

**End** **If**

' Yönetilmeyen kaynakların temizleneceği bölüm.

**End** **If**

disposed = **True**

**End** **Sub**

**Protected** **Overrides** **Sub** Finalize()

' Temizle metodu false değeri ile çağrılarak, GC tarafından bu çağrımın

' yapıldığı belirtilmektedir.

Temizle(**False**)

**End** **Sub**

**End** **Class**

Kod 110, Sonlandırılabilme ve Yok Edilebilme Örnek Şablonu

Tip Dönüşümleri

Bu konu başlığında, kitabın önceki konularında aktarılan tip dönüşümleri hakkında daha detaylı ve ileri seviye bilgiler verilecektir.

Tip dönüşümleri dolaylı (ing. Implicit) ve direkt (ing. Explicit) olarak yapılabilir. Nümerik tip dönüşüm işlemlerinde, küçük değer aralığın büyük değer aralığına dönüşümü dolaylı olarak derleyici tarafından yapılabilir. Örneğin byte tipinde bir değer dolaylı olarak (otomatik olarak derleyici tarafından) integer tipe dönüştürülebilir. Tersi durumlarda, tip dönüşüm kuralları ile direkt dönüşüm programcı tarafından yapılmalıdır.

| C# | VB.NET |
| --- | --- |
| **int** j = 12;  ' Dolaylı tip dönüşüm işlemi  **long** l = j;  // Direkt tip dönüşüm işlemi  **int** i = (**int**)l; | **Dim** j **As** **Integer** = 12  ' Dolaylı tip dönüşüm işlemi  **Dim** l **As** **Long** = j  ' Direkt tip dönüşüm işlemi  **Dim** i **As** **Integer** = **CType**(l, **Integer**) |

Kod 111, Dolaylı ve Dolaysız Nümerik Tip Dönüşümleri

Sınıflar için tip dönüşümü benzer şekilde yapılmaktadır. Polimorfizm kuralları çerçevesinde, miras alan nesne dolaylı olarak ata sınıf nesnesine dönüştürülebilir. Tersi durumda, ata sınıf tipinde tanımlanan bir nesnenin miras alan nesneye dönüşümünde direkt dönüşüm yapılmalıdır.

| C# | VB.NET |
| --- | --- |
| **class** Ata {  }  **class** MirasAlan : Ata {  }  // Dolaylı tip dönüşümü işlemi  Ata a = **new** MirasAlan();  // Direkt tip dönüşüm işlemi  MirasAlan m = (MirasAlan)a; | **Class** Ata  **End** **Class**  **Class** MirasAlan  **Inherits** Ata  **End** **Class**  ' Dolaylı tip dönüşümü işlemi  **Dim** a **As** Ata = **New** MirasAlan  ' Direkt tip dönüşüm işlemi  **Dim** m **As** MirasAlan  m = **CType**(a, MirasAlan) |

Kod 112, Dolaylı ve Dolaysız Sınıf Tip Dönüşümleri

Ata sınıf – miras alan sınıf hiyerarşik yapısı kapsamında yapılabilen tip dönüşümlerinin, birbirleri ile ilgisi olmayan sınıflar arasında da yapılması mümkündür.

Dolaylı tip dönüşüm işlemi derleyici tarafından otomatik olarak yapılmaktadır. Dolaylı ve direkt tip dönüşüm işlemlerinin farklı, dolaylı dönüşüm işlemi ile değerlerin birbirine dönüşümünde, derleme anında dönüşüm işleminin yapılabilirliğinin güvence altında olması, direkt dönüşüm işleminde ise çalışma anında dönüşümün yapılmasıdır. Çalışma anında tip dönüşüm işlemi başarısız olursa InvalidCast hatası üretilecektir.

Özelleştirilmiş Tip Dönüşümleri

Farklı (birbirleri ile hiyerarşik ilişkisi bulunmayan) sınıfların veya yapıların birbirlerine dönüşüm işlemini özelleştirmek mümkündür.

| C# | VB.NET |
| --- | --- |
| **class** Daire {  **private** **int** r;  **public** Daire(**int** r) {  **this**.r = r;  }  }  **class** Elips {  **private** **int** r1;  **private** **int** r2;  **public** Elips(**int** r1, **int** r2) {  **this**.r1 = r1;  **this**.r2 = r2;  }  }  // Kullanım Örnekleri:  Elips e = **new** Elips(5, 3);  // Geçersiz tip dönüşüm örnekleri // aşağıda gösterilmiştir.  // Sınıfların hiyerarşik olarak // ilişkileri olmadığı için gerek // dolaylı gerek dolaysız tip  // dönüşüm işlemleri derleme zamanı // hatası alacaktır.  Daire d = e;  Daire d = (Daire)e; | **Class** Daire  **Private** r **As** **Integer**  **Public** **Sub** **New**(**ByVal** r **As** **Integer**)  **Me**.r = r  **End** **Sub**  **End** **Class**  **Class** Elips  **Private** r1 **As** **Integer**  **Private** r2 **As** **Integer**  **Public** **Sub** **New**(**ByVal** r1 **As** **Integer**,   **ByVal** r2 **As** **Integer**)  **Me**.r1 = r1  **Me**.r2 = r2  **End** **Sub**  **End** **Class**  ' Kullanım Örnekleri:  **Dim** e **As** Elips = **New** Elips(5, 3)  ' Geçersiz tip dönüşüm örnekleri ' aşağıda gösterilmiştir.  ' Sınıfların hiyerarşik olarak ' ilişkileri olmadığı için gerek ' dolaylı gerek dolaysız tip  ' dönüşüm işlemleri derleme zamanı ' hatası alacaktır.  **Dim** d **As** Daire = e  **Dim** d **As** Daire = **CType**(e, Daire) |

Özelleştirilmiş Dolaylı Tip Dönüşümleri

Implicit ve Widening (sırasıyla C# ve VB.NET) dolaylı tip dönüşüm işleminin sınıf ve yapılar için özelleştirilebilmesini sağlayan rezerve kelimelerdir.

C#

**class** Elips {

. . .

**public** **static** **implicit** **operator** Daire(Elips e) {

// Dolaylı tip dönüşümü yapıldığında otomatik olarak bu metot   
 // çalıştırılacaktır. Metodun Daire tipinde bir değer döndürdüğüne ve Elips

// tipinde bir değeri parametre olarak aldığına dikkat ediniz.

**return** **new** Daire(System.Math.Min(e.r1, e.r2));

}

}

// Kullanım Örnekleri:

Elips e = **new** Elips(5, 3);

// Aşağıdaki kullanım geçerlidir ve 3 çapında bir daire nesnesine dönüşüm   
// yapılmaktadır.

daire d = e;

VB.NET

**Class** Elips

. . .

**Public** **Shared** **Widening** **Operator** **CType**(**ByVal** e **As** Elips) **As** Daire

' Dolaylı tip dönüşümü yapıldığında otomatik olarak bu metot   
 ' çalıştırılacaktır. Metodun Daire tipinde bir değer döndürdüğüne ve Elips

' tipinde bir değeri parametre olarak aldığına dikkat ediniz.

**Return** **New** Daire(System.Math.Min(e.r1, e.r2))

**End** **Operator**

**End** **Class**

' Kullanım Örnekleri:

**Dim** e **As** Elips = **New** Elips(5, 3)

' Aşağıdaki kullanım geçerlidir ve 3 çapında bir daire nesnesine

' dönüşüm yapılmaktadır.

**Dim** d **As** Daire = e

Kod 113, Özelleştirilmiş Dolaylı Tip Dönüşüm Örneği

Örnekte, dolaylı tip dönüşümünde otomatik olarak çalıştırılan metodun statik olarak tanımlandığına dikkat ediniz.

Özelleştirilmiş Direkt Tip Dönüşümleri

Explicit ve Narrowing (sırasıyla C# ve VB.NET) direkt tip dönüşüm işleminin sınıf ve yapılar için özelleştirilebilmesini sağlayan rezerve kelimelerdir.

C#

**class** Elips {

. . .

**public** **static** **explicit** **operator** Daire(Elips e) {

**return** **new** Daire(System.Math.Min(e.r1, e.r2));

}

}

// Kullanım Örnekleri:

Elips e = **new** Elips(5, 3);

// Aşağıdaki kullanım geçerlidir ve 3 çapında bir daire nesnesine

// dönüşüm yapılmaktadır.

Daire d = (Daire)e;

VB.NET

**Class** Elips

. . .

**Public** **Shared** **Narrowing** **Operator** **CType**(**ByVal** e **As** Elips) **As** Daire

**Return** **New** Daire(System.Math.Min(e.r1, e.r2) )

**End** **Operator**

**End** **Class**

' Kullanım Örnekleri:

**Dim** e **As** Elips = **New** Elips(5, 3)

' Aşağıdaki kullanım geçerlidir ve 3 çapında bir daire nesnesine

' dönüşüm yapılmaktadır.

**Dim** d **As** Daire = **CType**(e, Daire)

Kod 114, Özelleştirilmiş Direkt Tip Dönüşüm Örneği

Örnekte, direkt tip dönüşümünde otomatik olarak çalıştırılan metodun statik olarak tanımlandığına dikkat ediniz.

Bir sınıf içerisinde aynı tipi döndüren ve aynı parametre listesine sahip dolaylı ve direkt dönüşüm metotları birlikte tanımlanamaz.

Operatör Yükleme

Tip dönüşümünün özelleştirilmesine benzer şekilde, .NET içerisinde operatörlerin de sınıf ve yapı tipleri için özelleştirilebilmesi mümkündür.

Esas tipler için (Tamsayı, karakter dizi vb.) kitabın daha önceki bölümlerinde kullanımları ve anlamları aktarılan operatörler, sınıf ve yapı tipleri için özelleştirilerek daha temiz ve okunabilir kaynak kod yapısı sağlanabilir.

Örneğin, yukarıdaki konu içerisinde verilen Daire ve Elips sınıfları için, Daire tipinde iki nesnenin toplanması ile (örneğin + operatörü ile) daha büyük çapa sahip olan yeni bir Daire nesnesi oluşturulabilir.

Aşağıda operatör yükleme ile ilgili çeşitli örnekler gösterilmiştir.

C#

**class** Daire {

**private** **int** r;

**public** Daire(**int** r) {

**this**.r = r;

}

// + operatör yükleme işlemi

**public** **static** Daire **operator** +(Daire d1, Daire d2) {

**return** **new** Daire(d1.r + d2.r);

}

// == operatör yükleme işlemi

**public** **static** **bool** **operator** == (Daire d1, Daire d2) {

**return** d1.r == d2.r;

}

// != operatör yükleme işlemi. Eşitlik operatörünün yüklenmesi

// durumunda eşitsizlik operatörü de yüklenmelidir.

**public** **static** **bool** **operator** != (Daire d1, Daire d2) {

**return** d1.r != d2.r;

}

// < operatör yükleme işlemi. Küçüktür operatörünün yüklenmesi

// durumunda büyüktür operatörü de yüklenmelidir.

**public** **static** **bool** **operator** < (Daire d1, Daire d2) {

**return** d1.r < d2.r;

}

// > operatör yükleme işlemi.

**public** **static** **bool** **operator** > (Daire d1, Daire d2) {

**return** d1.r > d2.r;

}

}

**static** **class** Program {

**static** **void** Main(**string**[] args) {

Daire d1 = **new** Daire(4);

Daire d2 = **new** Daire(3);

// d1 + d2 anlatım değerinin hesaplanmasında Daire sınıfı

// tarafından sağlanan + operatör yükleme kodu otomatik olarak

// çalışmaktadır.

Daire d3 = d1 + d2;

Daire d4 = **new** Daire(4);

// Referanslar birbirine eşitlenmektedir. Böylece d5 isimli referans ve d2   
 // isimli referans aynı nesneyi göstermektedirler.

Daire d5 = d2;

Console.WriteLine("d4 EŞİT d1: {0}", d4 == d1);

Console.WriteLine("d5 EŞİT d2: {0}", d5 == d2);

Console.WriteLine("d3 EŞİT d1 + d2: {0}", d3 == d1 + d2);

Console.WriteLine("d4 REF EŞİT d1: {0}", d4.Equals(d1));

Console.WriteLine("d5 REF EŞİT d2: {0}", d5.Equals(d2));

Console.WriteLine("d2 BÜYÜK d1: {0}", d2 > d1);

Console.WriteLine("d3 KÜÇÜK d1: {0}", d3 < d1);

}

}

VB.NET

**Class** Daire

**Private** r **As** **Integer**

**Public** **Sub** **New**(**ByVal** r **As** **Integer**)

**Me**.r = r

**End** **Sub**

' + operatör yükleme işlemi

**Public** **Shared** **Operator** + (**ByVal** d1 **As** Daire, **ByVal** d2 **As** Daire) **As** Daire

**Return** **New** Daire(d1.r + d2.r)

**End** **Operator**

' == operatör yükleme işlemi

**Public** **Shared** **Operator** = (**ByVal** d1 **As** Daire, **ByVal** d2 **As** Daire) **As** **Boolean**

**Return** d1.r = d2.r

**End** **Operator**

// <> operatör yükleme işlemi. Eşitlik operatörünün yüklenmesi durumunda   
 // eşitsizlik operatörü de yüklenmelidir.

**Public** **Shared** **Operator** <> (**ByVal** d1 **As** Daire, **ByVal** d2 **As** Daire) **As** **Boolean**

**Return** d1.r <> d2.r

**End** **Operator**

' < operatör yükleme işlemi. Küçüktür operatörünün yüklenmesi durumunda   
 ' büyüktür operatörü de yüklenmelidir.

**Public** **Shared** **Operator** < (**ByVal** d1 **As** Daire, **ByVal** d2 **As** Daire) **As** **Boolean**

**Return** d1.r < d2.r

**End** **Operator**

' > operatör yükleme işlemi.

**Public** **Shared** **Operator** > (**ByVal** d1 **As** Daire, **ByVal** d2 **As** Daire) **As** **Boolean**

**Return** d1.r > d2.r

**End** **Operator**

**End** **Class**

**Class** Program

**Shared** **Sub** Main(**ByVal** args() **As** **String**)

**Dim** d1 **As** Daire = **New** Daire(4)

**Dim** d2 **As** Daire = **New** Daire(3)

' d1 + d2 anlatım değerinin hesaplanmasında Daire sınıfı tarafından   
 ' sağlanan + operatör yükleme kodu otomatik olarak çalışmaktadır.

**Dim** d3 **As** Daire = d1 + d2

**Dim** d4 **As** Daire = **New** Daire(4)

' Referanslar birbirine eşitlenmektedir. Böylece d5 isimli referans ve d2   
 ' isimli referans aynı nesneyi göstermektedirler.

**Dim** d5 **As** Daire = d2

Console.WriteLine("d4 EŞİT d1: {0}", d4 = d1)

Console.WriteLine("d5 EŞİT d2: {0}", d5 = d2)

Console.WriteLine("d3 EŞİT d1 + d2: {0}", d3 = d1 + d2)

Console.WriteLine("d4 REF EŞİT d1: {0}", d4.Equals(d1))

Console.WriteLine("d5 REF EŞİT d2: {0}", d5.Equals(d2))

Console.WriteLine("d2 BÜYÜK d1: {0}", d2 > d1)

Console.WriteLine("d3 KÜÇÜK d1: {0}", d3 < d1)

**End** **Sub**

**End** **Class**

Kod 115, Operatör Yükleme Örnekleri

Kod çıktısı aşağıda gösterilmiştir.

d4 EŞİT d1: True

d5 EŞİT d2: True

d3 EŞİT d1 + d2: True

d4 REF EŞİT d1: False

d5 REF EŞİT d2: True

d2 BÜYÜK d1: False

d3 KÜÇÜK d1: False

Yukarıdaki kod ve çıktı değerlendirildiğinde, Object.Equals metodunun sadece referans edilen nesnelerin aynı olması durumunda True değerini döndürdüğü görülecektir. Örnekte, d5 ve d2 isimli değişkenler ile ifade edilen (işaret edilen veya referans edilen) nesne aynı nesnedir. Bu durumda Object.Equals metodu True değerini döndürmüştür. Buna rağmen, d4 ve d1 isimli değişkenler ile ifade dilen nesneler farklı nesneler olmasına rağmen, Daire sınıfı eşitlik operatör yüklemesini yaptığı için, d4 == d1 anlatımı True olarak hesaplanmıştır.

Sanal bir metot olan Object.Equals metodu örtülerek, bu metodun referans eşitliği yerine değer eşitliği kontrolü yapması sağlanabilir.

C#

**class** Daire {

**public** **override** **bool** Equals(**object** o) {

// Gelen değer Daire tipinde ise işleme başla.

**if** (o **is** Daire) {

Daire obj = (Daire)o;

// Parametre ile ifade edilen nesnenin çapı ile bu nesnenin çapı

// aynı ise True döndür.

**return** obj.r == **this**.r;

}

**return** **false**;

}

. . .

VB.NET

**Class** Daire

**Public** **Overrides** **Function** Equals(**ByVal** o **As** **Object**) **As** **Boolean**

' Gelen değer Daire tipinde ise işleme başla.

**If** **TypeOf** (o) **Is** Daire **Then**

**Dim** obj **As** Daire

obj = **CType**(o, Daire)

' Parametre ile ifade edilen nesnenin çapı ile bu nesnenin çapı

' aynı ise True döndür.

**Return** obj.r = **Me**.r

**End** **If**

**Return** **False**

**End** **Function**

. . .

Kod 116, Object.Equals Metodunun Örtülmesi

Bir sınıf için eşitlik operatörünün yüklenmesi veya Equals metodunun örtülmesi durumunda Object.GetHashCode metodunun da örtülmesi önerilmektedir.

Object.GetHashCode metodunun temel amacı, nesnenin o anki durumuna göre tekil bir değer üretmektir. Örneklerde, Daire sınıfı için ayırt edici değer dairenin sahip olduğu çaptır. Dikkat edilirse çapı aynı iki daire nesnesi eşit kabul edilmiştir. Bu durumda, Daire sınıfı için çap bilgisi GetHashCode metodunda kullanılabilir.

C#

**class** Daire {

. . .

**public** **override** **int** GetHashCode() {

**return** ToString().GetHashCode();

}

**public** **override** **string** ToString() {

**return** **string**.Format("Daire: {0}", r);

}

. . .

VB.NET

**Class** Daire

. . .

**Public** **Overrides** **Function** GetHashCode() **As** **Integer**

**Return** ToString().GetHashCode()

**End** **Function**

**Public** **Overrides** **Function** ToString() **As** **String**

**Return** **String**.Format("Daire: {0}", r)

**End** **Function**

. . .

Kod 117, Object.GetHashCode Kullanım Örneği

GetHashCode metodu özet bilgi (ing. Hash) bazlı listelere eleman eklendiğinde çağrılmaktadır. Bu konuyla ilgili kitabın ilerleyen bölümlerinde bilgiler verilecektir.

Listelenen kaynak kodlar \Bölüm 6\Opload klasöründe bulunmaktadır.

Bu bölümün tamamlanması ile okuyucuya teorik düzeyde önemli bilgiler sunulmuştur. Kitabın ilerleyen bölümlerinde bu bilgilerin kullanımı, somut örnekler ve pratik uygulama yöntemleri ile birlikte aktarılıyor olacaktır.  
  
.NET programlama konusunda uzmanlaşmak isteyen her programcının verilen bilgileri iyi düzeyde anlamış ve uygulayabiliyor olması büyük önem arz etmektedir.  
  
.NET konusuna ilk defa bu kitap ile başlayan ve daha önce bu konuda çalışma yapmamış olan programcılara, konunun genelini ve amacını tam olarak anlamış olmaları, buna rağmen detayda anlayamadıkları bölümler üzerinde zaman kaybetmemeleri önerilmektedir. Bir çok örnek kullanım ile desteklenecek olan teorik konuların kitabın ilerleyen bölümlerinin de okunması ile daha iyi anlaşılacağı değerlendirilmektedir.

Üç türlü aristokrasi vardır; birincisi yaş ve kıdem; ikincisi servet; üçüncüsü akıl ve bilgidir. En şereflisi sonuncusudur.

Schopenhauer

Bölüm

7

İstisna Yönetimi

K

aliteli yazılım kavramından anlaşılabilecek önemli konulardan birisi, yazılım tarafından yapılan hata yönetimi ve hatalar karşısında yazılımın verdiği tepkidir.

Bu bölümde, .NET içerisinde istisnai durumların ne şekilde yönetilebileceği ve kavramsal düzeyde istisnalar hakkında bilgiler verilecektir.

Temel Bilgiler ve Öneriler

Derleyici, derleme zamanında oluşan hataları ve kaynak kod içerisinde gördüğü potansiyel uyumsuzlukları uyarı olarak vermektedir. Buna rağmen çalışma zamanında oluşan hataların ne şekilde ele alınacağından programcı sorumludur.

Çalışma zamanında farklı şekillerde hatalar oluşabilmektedir. Örneğin uygulama yazılımının okumaya çalıştığı bir dosya disk üzerinde bulunmayabilir veya kullanıcının girdi bilgilerinden kaynaklı veri tutarsızlıkları ortaya çıkabilir. Bu gibi durumlarda, gerekli istisna yönetimi programcı tarafından yapılmamış ise, .NET tarafından hata mesajı gösterilir ve bazı durumlarda uygulama çalışmaya devam edemez.

İstisna (ing. Exception), çalışma zamanında oluşan hata olarak tanımlanabilir. Kontrol edilmeyen bir istisna, yazılım kalitesini düşürebilir, önemli güvenlik açıklarına yol açabilir.

İstisna yönetimi ile ilgili aşağıda çeşitli öneriler listelenmiştir.

* İstisna yönetimi performans açısından negatif etkiye sahiptir. Bu yüzden sadece gerektiği yerde gerektiği kadar istisna yönetimi yapılmalıdır. Örneğin bir metot içerisinden farklı metotlar çağrılabilir. Her metot içerisinde istisna yönetiminin yapılması performans açısından olumsuz durumlar oluşturabilir.
* Teknik (dosyanın okunamaması, veritabanı bağlantısının yapılamaması gibi kullanıcıdan kaynaklanmayan) istisnaların tüm detayları ile gösterimi önemli güvenlik açıklarına yol açabilir. Kötü niyetli bir kullanıcı hata mesajından yola çıkarak sistem hakkında bilgiler alabilir.
* Özellikle teknik istisnaların gerektiğinde incelenmek üzere tüm detayları ile saklanması önerilmektedir. Oluşan istisnalar sistem olay günlüğüne (ing. Event Log) yazılabilir. İstisnanın saklanması veya oluştuğunda farklı yöntemler ile ilgililere iletimi istisnanın yayınlanması (ing. Exception Publishing) olarak ifade edilmektedir.

Konunun ilerleyen bölümlerinde, listelenen önerilerin .NET içerisinde ne şekilde ele alınabileceği ve uygulanabileceği hakkında bilgiler sunulacaktır.

İstisna Türleri

Temel olarak iki istisna türünden bahsetmek mümkündür. Bunlar kullanıcı kaynaklı istisnalar ve teknik istisnalar olarak tanımlanabilir.

Kullanıcı kaynaklı istisnalar, kullanıcının yaptığı hatalı bir işlem sonucu ortaya çıkan istisnalardır. Uygulama yazılımı tarafından sadece 0 – 100 aralığında veri beklenirken kullanıcının bu aralık dışında veri girmesi kullanıcı kaynaklı istisnalara örnek olarak verilebilir. Benzer şekilde kullanıcının yetkisi olmayan bir işlemi yapmaya çalışması da, güvenlik ile ilgili olmasına rağmen, kullanıcı kaynaklı bir hata olarak değerlendirilebilir.

Teknik istisnalar, yazılımda olan hatalar (ing. Bug) veya yazılımın dağıtımından kaynaklanabilecek bir problemden ortaya çıkabilen, kullanıcıdan kaynaklanmayan istisnalardır. Veritabanına bağlanmaya çalışıldığında bağlanamama veya bozuk bir dosyayı okuyamama gibi işlemler teknik istisnalara örnek olarak verilebilir.

Temel İstisna Sınıfı

İstisna, .NET içerisinde System.Exception sınıfı ile ifade edilmektedir. System.Exception sınıfı, istisna hakkında çeşitli bilgilere ulaşabilmeyi sağlayan özellik ve metotlar içermektedir.

Sistem İstisnaları

System.SystemException sınıfı CLR tarafından üretilen ve uygulamanın çalışmaya devam etmesini etkileyebilecek istisnaları ifade etmektedir.

SystemException sınıfı System.Exception sınıfını miras almaktadır. Bu sınıf ve bu sınıfı miras alan farklı sınıflar .NET içerisinde tanımlanmıştır.

Örneğin System.IO ad alanı içerisinde girdi – çıktı işlemlerinde oluşabilecek istisnaları belirtir (FileNotFoundException, IOException, vb.), System.Data içerisinde veri işlemlerinde olabilecek istisnaları belirtir (DataException, ConstraintException, vb.) farklı istisna sınıfı tanımları bulunmaktadır. .NET sınıf kütüphanesi içerisinde tanımlı tüm istisnalar SystemException sınıfını miras almaktadır.

SystemException sınıfının tanımlanma amacı, çalışma anında oluşabilecek bir istisnanın .NET içerisinden kaynaklandığını belirtmektir. Bunun dışında sınıf içerisinde herhangi özel kod gerçekleştirimi bulunmamaktadır.

SystemException sınıfını miras alan çeşitli istisna sınıfları ve açıklamaları aşağıda gösterilmiştir.

| Sınıf | Açıklama |
| --- | --- |
| System.OutOfMemoryException | Hafıza alanının işlem için yetersiz olduğu durumu belirten istisna. |
| System.InvalidCastException | Geçersiz tip dönüşüm işlemini belirten istisna. |
| System.FormatException | İletilen karakter dizi format değerinin geçersiz olduğunu belirten istisna. |
| System.NullReferenceException | Değeri null olan bir referansa erişimi belirten istisna. |
| System.IO.FileNotFoundException | Açılmak istenen dosyanın disk üzerinde bulunamadığını belirten istisna. |
| System.Net.WebException | Ağ (ing. Network) ile ilgili bir hata olduğunu belirten istisna. |
| System.Security.SecurityException | Güvenlik ihlalini belirten istisna. |
| System.Xml.XmlException | XML ile ilgili bir hata olduğunu belirten istisna. Bu sınıfı miras alan ve daha detaylı bilgiye ulaşımı sağlayan istisnalar System.Xml ad alanı içerisinde bulunmaktadır. |

Tablo 42, Çeşitli SystemException İstisnaları ve Açıklamaları

SystemException sınıfının uygulamalar içerisinde direkt veya miras alınarak kullanımı önerilmemektedir.  
  
İstisna sınıf isimlerinin Exception ile sonlanması bir zorunluluk olmamasına rağmen önerilen kullanım yöntemidir.

Uygulama İstisnaları

System.ApplicationException sınıfı, uygulama yazılımlarında kullanımı önerilen ve istisnanın .NET sınıf kütüphanesinden gelmediğini belirten sınıftır. Uygulama yazılımlarında direkt ApplicationException sınıfı kullanılabileceği gibi bu sınıfı miras alan farklı sınıflar da kullanılabilir.

İstisna Yönetim Modeli

Mimarisi ve niteliği ne olursa olsun tüm uygulama yazılımlarında istisna yönetim modeli uygulamak mümkündür. Yukarıda aktarılan istisna türleri ve öneriler doğrultusunda belirlenen, mantıksal düzeyde tüm yazılımlarda kullanımı önerilen istisna yönetim modeli aşağıda gösterilmiştir.

Şekil 19, Önerilen İstisna Modeli

Önerilen istisna yönetim modelinde, System.ApplicationException sınıfını miras alan kullanıcı kaynaklı ve teknik istisnaları belirten iki yeni sınıf tanımı bulunmaktadır. Bu sınıfların programcı tarafından tanımlanması öngörülmektedir.

Kullanıcı kaynaklı hataları temelde ikiye ayırmak mümkündür. Güvenlik hataları, kullanıcının yetkisi olmayan işlemlere erişmeye çalışmasından kaynaklı hatalardır. Veri doğrulama hataları, kullanıcının girdiği geçersiz bilgilerden kaynaklanan hatalardır. Kullanıcı kaynaklı hataların tek bir sınıf çatısı altında toplanması ile hata türleri, dolayısıyla tanımlanabilecek istisna türleri de çeşitlenebilir.

Teknik hataları tek bir sınıfla yönetmek veya bu sınıfı miras alan farklı sınıflar kullanmak mümkündür.

Model, yazılımda oluşan her türlü istisnayı yayınlayabilecek istisna yayınlayıcı sınıfı içermektedir. Bu sınıf yardımıyla, filtreleme özelliklerinin de eklenmesi ile, istenilen tipte olan istisnaların olay günlüğüne yazılması veya e-posta, SMS gibi yöntemlerle iletimi mümkün olmaktadır.

Modelin tüm yazılımlarda uygulanabilirliği, tekrar kullanılabilir sınıf kütüphanesi (ing. Class Library) olarak geliştirilebilmesini de olanaklı kılmaktadır.

Modelin kod gerçekleştirim detayları ile ilgili bilgiler kitabın ilerleyen bölümlerinde sunulacaktır.

İstisna Kullanım Bilgileri

Bu bölümde, yazılımda oluşan bir istisnanın yakalanması, istisna sınıf özellikleri ve istisna fırlatma konularında bilgiler verilecektir.

İstisna Yakalama

Try - Catch bloğu oluşan istisnanın yakalanması amacıyla kullanılmalıdır.

| C# | VB.NET |
| --- | --- |
| **try** {  // Korunumlu blok içerisinde  // yapılan kod gerçekleştirimi.  }  **catch** {  // try - catch arasında bulunan  // kaynak kod içerisinde herhangi  // bir hata oluştuğunda   // çalıştırılacak blok.  }  //--------------------------------//  **try** {  }  **catch** (Exception exc) {  // Korunumlu blok içerisinde   // oluşan istisna nesnesi exc ile  // ifade edilmektedir.  }  //--------------------------------//  **try** {  }  **catch** (EndOfStreamException exc) {  // Korunumlu blok içerisinde  // EndOfStreamException sınıfı   // veya bu sınıfı miras alan başka  // bir sınıf tipinde istisna   // oluşması durumunda   // çalıştırılacak blok.  }  **catch** (Exception exc) {  // EndOfStreamException dışındaki  // istisnalar için çalıştırılacak  // blok.  }  //--------------------------------//  **try** {  }  **catch** (EndOfStreamException) {  // Sadece EndOfStreamException  // istisnası oluşması durumunda  // çalıştırılacak blok.  } | **Try**  ' Korunumlu blok içerisinde  ' yapılan kod gerçekleştirimi.  **Catch**  ' try - catch arasında bulunan  ' kaynak kod içerisinde herhangi  ' bir hata oluştuğunda   ' çalıştırılacak blok.  **End** **Try**  **Try**  **Catch** exc **As** Exception  ' Korunumlu blok içerisinde   ' oluşan istisna nesnesi exc ile  ' ifade edilmektedir.  **End** **Try**  **Try**  **Catch** exc **As** EndOfStreamException  ' Korunumlu blok içerisinde  ' EndOfStreamException sınıfı   ' veya bu sınıfı miras alan başka  ' bir sınıf tipinde istisna   ' oluşması durumunda   ' çalıştırılacak blok.  **Catch** exc **As** Exception  ' EndOfStreamException dışındaki  ' istisnalar için çalıştırılacak  ' blok.  **End** **Try**  **Dim** i **As** **Integer** = 5  **Try**  **Catch** **When** i = 4  ' Korunumlu blok içerisinde  ' içerisinde istisna oluşması  ' ve I isimli değişken değeri  ' 4 ise çalıştırılacak blok.  **Catch When** i <> 3  ' I değeri 3 değilse   ' çalıştırılacak blok.  **End** **Try**  **Try**  **Catch** exc **As** EndOfStreamException   **When** i = 1  ' İstisna EndOfStreamException   ' tipinde ve I isimli değişken  ' değeri 1 ise çalıştırılacak   ' blok.  **End** **Try** |

Kod 118, İstisna Yakalama Örnekleri

Finally Bloğu

Bazı durumlarda, korunumlu blok içerisinde istisna oluşup oluşmamasına bakmazsızın mutlaka çalıştırılması istenen kodlar olabilir. Örneğin bir dosyanın yazılım tarafından açılması ardından, dosyanın okunması sırasında bir hata oluşsa dahi dosyanın kapatılması gereklidir. Bu nitelikte bir gereksinimde try – finally bloğu kullanılabilir.

| C# | VB.NET |
| --- | --- |
| **try** {  // Korunumlu blok.  }  **finally** {  // Korunumlu blok tamamlandıktan   // sonra, istisna oluşsa dahi,   // çalıştırılacak blok.  }  // try – finally bloğunun istisna // kontrolü ile kullanımı. Aşağıdaki // yapıda korunumlu blok içerisinde // istisna oluşmazsa sadece finally // bloğu çalıştırılacaktır. İstisna // oluşması durumunda catch bloğu // çalıştırılacak, ardından finally // bloğu çalıştırılacaktır.  **try** {  // Korunumlu blok.  }  **catch** (Exception exc) {  // İstisna oluşması durumunda  // çalıştırılacak blok.  }  **finally** {  // Her durumda çalıştırılacak   // blok.  } | **Try**  ' Korunumlu blok.  **Finally**  ' Korunumlu blok tamamlandıktan   ' sonra, istisna oluşsa dahi,   ' çalıştırılacak blok.  **End** **Try**  ' try – finally bloğunun istisna ' kontrolü ile kullanımı. Aşağıdaki ' yapıda korunumlu blok içerisinde ' istisna oluşmazsa sadece finally ' bloğu çalıştırılacaktır. İstisna ' oluşması durumunda catch bloğu ' çalıştırılacak, ardından finally ' bloğu çalıştırılacaktır.  **Try**  ' Korunumlu blok.  **Catch** exc **As** Exception  ' İstisna oluşması durumunda  ' çalıştırılacak blok.  **Finally**  ' Her durumda çalıştırılacak   ' blok.  **End** **Try** |

Kod 119, Finally Bloğu Kullanım Örnekleri

İstisna Fırlatma

İstisna fırlatma (ing. Exception Throwing), işlem sırasında beklenilmeyen bir durum oluştuğunda ve oluşan durumun işlemin devam etmesini engelleyici nitelikte olduğunda kullanılan yöntemdir.

Örneğin, veritabanına bilgilerin kaydedilmesi sırasında aşağıdaki işlemler sırasıyla yazılım tarafından yapılmalıdır.

[1] Veritabanı bağlantı dizesini oku

[2] Veritabanı bağlantısı yap

[3] SQL cümlesini çalıştır

[4] Veri tabanı bağlantısını kapat

2 numaralı işlem sırasında bir hata oluşması durumunda SQL cümlesinin çalıştırılması anlamsızdır. Böyle bir durumda uygulama istisna fırlatarak işlemin başarısız olduğunu belirtmelidir.

İstisna fırlatma, rezerve bir kelime olan throw ile yapılmaktadır. İstisna fırlatma kavramı, herhangi istisna nesnesinin (System.Exception tipinde) anlam kazanması olarak değerlendirilebilir.

C#

**static** **string** DosyaOku(**string** dosya) {

**if** (**string**.IsNullOrEmpty(dosya))

**throw** **new** Exception("Dosya adı geçersiz");

**if** (!File.Exists(dosya))

**throw** **new** Exception("Dosya bulunamadı");

. . .

}

VB.NET

**Shared** **Function** DosyaOku(**ByVal** dosya **As** **String**) **As** **String**

**If** **String**.IsNullOrEmpty(dosya) **Then**

**Throw** **New** Exception("Dosya adı geçersiz")

**End** **If**

**If** **Not** File.Exists(dosya) **Then**

**Throw** **New** Exception("Dosya bulunamadı")

**End** **If**

. . .

**End** **Function**

Kod 120, İstisna Fırlatma Örneği

Aşağıda istisna yakalama ile birlikte örnek kullanım gösterilmiştir.

| C# | VB.NET |
| --- | --- |
| **static** **void** Main() {  // Bu bölümde herhangi bir istisna  // kontrolü yapılmadığına ve  // aşağıdaki kodların try bloğu   // içerisine alınmadığına   // dikkat edilmelidir. İstisna  // kontrolünün sadece gerektiği  // durumlarda yapılması en doğru  // yaklaşımdır.  Console.WriteLine("Başlangıç");  Console.WriteLine("Dosya adı:");  **string** dosya = Console.ReadLine();  **try** {  // Korunumlu blok içerisinden  // DosyaOku metodu   // çağrılmaktadır. Bu metodun  // istisna fırlatmaması, dosya  // içeriğinin başarıyla  // okunduğu anlamına   // gelmektedir.  **string** s = DosyaOku(dosya);  // İstisna oluşmaması durumunda  // aşağıdaki kodlar   // çalıştırılacaktır ve catch  // bloğu çalıştırılmayacaktır.  // DosyaOku içerisinde istisna  // oluşması durumunda direkt  // catch bloğuna geçilecektir.  Console.WriteLine("İçerik:");  Console.WriteLine(s);  }  **catch** (Exception exc) {  // İstisna oluşması durumunda  // hata mesajını yaz.  Console.WriteLine(exc.Message);  }  **finally** {  // Her durumda çalıştırılacak  // blok.  Console.WriteLine("Bitiş");  }  } | **Shared** **Sub** Main()  ' Bu bölümde herhangi bir istisna  ' kontrolü yapılmadığına ve  ' aşağıdaki kodların try bloğu   ' içerisine alınmadığına   ' dikkat edilmelidir. İstisna  ' kontrolünün sadece gerektiği  ' durumlarda yapılması en doğru  ' yaklaşımdır.  Console.WriteLine("Başlangıç:")  Console.WriteLine("Dosya adı:")  **Dim** dosya **As** **String** =Console.ReadLine()  **Try**  ' Korunumlu blok içerisinden  ' DosyaOku metodu   ' çağrılmaktadır. Bu metodun  ' istisna fırlatmaması, dosya  ' içeriğinin başarıyla  ' okunduğu anlamına   ' gelmektedir.  **Dim** s **As** **String** = DosyaOku(dosya)  ' İstisna oluşmaması durumunda  ' aşağıdaki kodlar   ' çalıştırılacaktır ve catch  ' bloğu çalıştırılmayacaktır.  ' DosyaOku içerisinde istisna  ' oluşması durumunda direkt  ' catch bloğuna geçilecektir.  Console.WriteLine("İçerik:")  Console.WriteLine(s)  **Catch** exc **As** Exception  ' İstisna oluşması durumunda  ' hata mesajını yaz.  Console.WriteLine(exc.Message)  **Finally**  ' Her durumda çalıştırılacak  ' blok.  Console.WriteLine("Bitiş")  **End** **Try**  **End** **Sub** |

Kod 121, İstisna Fırlatma ve Yakalama Örneği

Örnek değerlendirildiğinde, istisna oluşması durumunda dosya içeriğinin ekrana yazılması yerine hata mesajı kullanıcıya gösterilmektedir.

Örnekte kullanılan yöntem, istisna niteliği ve sayısının az olduğu projelerde yeterli bir kontrol olarak değerlendirilse de, daha karmaşık ve büyük yazılım projelerinde hatanın tipi ve hata ile ilgili detaylı bilgilere erişmek gerekebilir. Örnekte, dosya adının geçersiz olması ile dosyanın bulunamaması durumunda Exception sınıfında bir istisna nesnesi fırlatılmaktadır. Hatayı oluşturan sebepler farklı olmasına rağmen, nitelikleri birbirinden farklı olan her iki hata için aynı istisna sınıfının kullanımı hataların birbirleri ile ayırt edilebilmesini zorlaştırmaktadır.

Kitabın ilerleyen bölümlerinde kullanıcı tanımlı istisna sınıfları hakkında daha detaylı bilgiler verilecektir.

İstisnanın Tekrar Fırlatılması

Bazı durumlarda yakalanan istisnanın tekrar fırlatılması (ing. Rethrowing Exception) gerekebilir. Örneğin istisna A isimli metot içerisinde catch bloğu ile yakalanarak bazı işlemler yapılması ve daha sonra istisnanın kaybedilmeden tekrar fırlatılması gerekebilir. Böylece, A metodu çağrımını yapan metot içerisinde işlem sırasında istisna oluştuğu anlaşılabilecektir.

| C# | VB.NET |
| --- | --- |
| // İstisnanın fırlatıldığı metot.  **static** **void** Test() {  **throw** **new**   InvalidDataException("X");  }  **static** **void** Test1() {  **try** {  Test();  }  **catch** {  // İstisna oluştuğunda yapılması   // istenilen işlemler bu bölümde   // yapılabilir.  // İstisna throw kelimesi   // kullanılarak tekrar   // fırlatılmaktadır.  **throw**;  }  }  **static** **void** Test2() {  **try** {  Test();  }  **catch** (Exception exc) {  // İstisna oluştuğunda yapılması   // istenilen işlemler bu bölümde   // yapılabilir. exc ile ifade  // edilen istisna nesnesi bu   // bölümde kullanılabilir.  // İstisna throw kelimesi   // kullanılarak tekrar   // fırlatılmaktadır.  **throw**;  }  }  // Aşağıdaki örnek istisna  // yönetiminde sık yapılan bir // hatayı göstermektedir. // Kod başarıyla derlenebilmesine // rağmen bu yöntem ile istisnanın // tekrar fırlatılması // önerilmemektedir.  **static** **void** Test3() {  **try** {  Test();  }  **catch** (Exception exc) {  // Aşağıdaki kullanım ile istisna  // nesnesi StackTrace özelliği  // değişmektedir.  **throw** exc;  }  } | ' İstisnanın fırlatıldığı metot.  **Shared** **Sub** Test()  **Throw** **New** InvalidDataException("X")  **End** **Sub**  **Shared** **Sub** Test1()  **Try**  Test()  **Catch**  ' İstisna oluştuğunda yapılması   ' istenilen işlemler bu bölümde   ' yapılabilir.  ' İstisna throw kelimesi   ' kullanılarak tekrar   ' fırlatılmaktadır.  **Throw**  **End** **Try**  **End** **Sub**  **Shared** **Sub** Test2()  **Try**  Test()  **Catch** exc **As** Exception  ' İstisna oluştuğunda yapılması   ' istenilen işlemler bu bölümde   ' yapılabilir. exc ile ifade  ' edilen istisna nesnesi bu   ' bölümde kullanılabilir.  ' İstisna throw kelimesi   ' kullanılarak tekrar   ' fırlatılmaktadır.  **Throw**  **End** **Try**  **End** **Sub**  ' Aşağıdaki örnek istisna  ' yönetiminde sık yapılan bir ' hatayı göstermektedir. ' Kod başarıyla derlenebilmesine ' rağmen bu yöntem ile istisnanın ' tekrar fırlatılması ' önerilmemektedir.  **Shared** **Sub** Test3()  **Try**  Test()  **Catch** exc **As** Exception  **Throw** exc  **End** **Try**  **End** **Sub** |

Kod 122, İstisnanın Tekrar Fırlatılması

Bir istisnanın tekrar fırlatılması sadece throw kelimesi ile yapılmalı, throw kelimesine parametre olarak yakalanan istisna verilmemelidir.

Aşağıda, örnekte verilen Test2 isimli metodun Main metodundan çağrımı ile yakalanan istisnanın Exception.StackTrace özelliği gösterilmiştir.

at ConsoleApplication.Program.Test() ... Line 57

at ConsoleApplication.Program.Test2() ... Line 78

at ConsoleApplication.Program.Main() ... Line 86

Çıktı değerlendirildiğinde, çağrımın Main metodundan başladığı, bu metot içerisinden Test2 isimli metodun çağrıldığı, son olarak Test2 isimli metot içerisinden de Test isimli metodun çağrıldığı görülecektir. En üstte olan Test isimli metot istisnanın fırlatıldığı metottur.

Aynı işlemin Test3 isimli metot ile yapılması sonucu oluşan StackTrace aşağıdadır.

at ConsoleApplication.Program.Test3() ... Line 78

at ConsoleApplication.Program.Main() ... Line 86

Test3 isimli metot içerisindeki kullanım ile StackTrace temizlenmiş ve istisnanın Test isimli metot tarafından fırlatıldığı bilgisi kaybolmuştur.

StackTrace bilgisinin özellikle temizlenmesi istenilen durumlarda istisna, Test3 isimli metotta gösterilen kullanım yöntemi ile tekrar fırlatılabilir.

Listelenen kaynak kodlar \Bölüm 7\Rethrow klasöründe bulunmaktadır.

İstisna Sınıfı ve Kullanıcı Tanımlı İstisnalar

Exception sınıfı, oluşan istisna hakkında detaylı bilgilere erişebilmeyi sağlayan üyeler içermektedir. Bu sınıfı miras alan farklı istisna sınıflarının programcı tarafından geliştirilmesi ile daha düzenli istisna yönetimi yapılabilmektedir.

Exception sınıfı ve üyeleri ile kullanıcı tanımlı istisna sınıflarının tanımlanması ve kullanımı bu konu başlığında aktarılacaktır.

İstisna Sınıfı Özellikleri

Aşağıda System.Exception sınıfı üye bilgileri gösterilmiştir.

| Üye Adı | Açıklama |
| --- | --- |
| Data | IDictionary arabirim gerçekleştirimini yapan herhangi bir nesnenin atanabileceği değerdir. Bu özelliğin temel amacı istisna hakkında daha detaylı bilgileri saklayabilmektir. |
| InnerException | İstisnadan önce oluşan diğer istisnalara ulaşmak için kullanılan salt okunur özelliktir. |
| HelpLink | İstisna ile ilgili yardım adresini belirtir. |
| Message | İstisnanın mesajını belirtir salt okunur özelliktir. Bu özellik, yapıcı metoda iletilen karakter dizi değeridir. |
| Source | İstisnanın oluştuğu Assembly dosyasını belirtir salt okunur özelliktir. |
| StackTrace | İstisnanın oluşması öncesi hangi metotların çağrıldığını belirtir salt okunur özelliktir. |
| TargetSite | İstisnanın oluştuğu metot hakkında daha detaylı bilgilere ulaşabilmeyi sağlayan salt okunur özelliktir. |

Tablo 43, System.Exception Sınıfı Üye Bilgileri

**Yapıcı Metotlar**

Yapıcı metotların kullanımı ile ilgili örnek kaynak kod ve açıklamalar aşağıda gösterilmiştir.

| C# | VB.NET |
| --- | --- |
| Exception exc1;  Exception exc2;  Exception exc3;  // Parametre almayan yapıcı metot. // Bu metodun kullanımı  // önerilmemektedir.  exc1 = **new** Exception();  // Hata mesajını parametre alan // yapıcı metot kullanımı. İletilen // hata mesajı Exception.Message // özelliğinin döndürdüğü değerdir.  exc2 = **new** Exception("Hata mesajı");  // Hata mesajı ile birlikte iç // istisnanın iletildiği yapıcı // metot. İletilen iç istisna // Exception.InnerException özelliği // tarafından döndürülen değerdir.  exc3 = **new** Exception("Hata", exc2); | **Dim** exc1 **As** Exception  **Dim** exc2 **As** Exception  **Dim** exc3 **As** Exception  ' Parametre almayan yapıcı metot. ' Bu metodun kullanımı  ' önerilmemektedir.  exc1 = **New** Exception()  ' Hata mesajını parametre alan ' yapıcı metot kullanımı. İletilen ' hata mesajı Exception.Message ' özelliğinin döndürdüğü değerdir.  exc2 = **New** Exception("Hata mesajı")  ' Hata mesajı ile birlikte iç ' istisnanın iletildiği yapıcı ' metot. İletilen iç istisna ' Exception.InnerException özelliği ' tarafından döndürülen değerdir.  exc3 = **New** Exception("Hata", exc3) |

Kod 123, İstisna Sınıfı Yapıcı Metot Örnekleri

**Exception.Data Kullanımı**

Exception.Data özelliğinin temel kullanım amacı istisna hakkında daha detaylı bilgileri saklayabilmektir. Örneğin bir web uygulamasında istisnanın oluştuğu anda hangi IP adresinden bağlantı yapıldığı veya kullanıcının tarayıcısının ne olduğu gibi bilgileri bu özellik içerisinde saklamak mümkündür.

| C# | VB.NET |
| --- | --- |
| // Aşağıda yeni bir istisna nesnesi // oluşturulmakta ve Data özelliğine // anahtar – değer ikilisi  // eklenmektedir.  Exception exc;  exc = **new** Exception("Hak Hatası");  exc.Data.Add("IP", UserHostAddress);  // Oluşturulan istisna nesnesi // fırlatılmaktadır.  **throw** exc;  // Örnek Kullanım:  **try** {  . . .  } **catch** (Exception exc) {  **string** ip;  ip = exc.Data["IP"].ToString());  } | ' Aşağıda yeni bir istisna nesnesi ' oluşturulmakta ve Data özelliğine ' anahtar – değer ikilisi  ' eklenmektedir.  **Dim** exc **As** Exception  exc = **New** Exception("Hak Hatası")  exc.Data.Add("IP", UserHostAddress)  ' Oluşturulan istisna nesnesi ' fırlatılmaktadır.  **Throw** exc  ' Örnek Kullanım:  **Try**  **. . .**  **Catch** exc **As Exception**  **Dim** ip **As** **String**  ip = exc.Data("IP").ToString()  **End Try** |

Exception.Data özelliği herhangi bir istisna nesnesine istenilen sayıda değer ekleyebilmeyi, böylece istisna hakkında daha detaylı bilgilere erişebilmeyi sağlamaktadır. Bu işlev, istisna sınıfını miras alan yeni istisna sınıfları tanımlamak ve tanımlanan sınıfa eklenen yeni özelliklere istenilen değerleri atayarak da elde edilebilir. Yukarıdaki örnekte System.Exception sınıfını miras alan, örneğin MyException isminde bir sınıf ve bu sınıf için IP isminde bir özellik tanımlanarak da aynı işlev sağlanabilir.  
  
Birbirlerinden mantıksal olarak farklı hataların yönetimi için yeni istisna sınıfları tanımlanması, Exception.Data özelliğini istisna ile ilgili detay bilgilere erişmek için kullanılması önerilen yöntemdir.

**Exception.InnerException Kullanımı**

Bir istisnayı oluşturan sebepler, istisnanın oluşumu öncesinde oluşan farklı istisnalardan kaynaklanabilir. Benzer şekilde, oluşan bir istisna yakalandıktan sonra farklı bir istisna olarak fırlatılmak istenebilir.

InnerException özelliğinin temel kullanım amaçlarından birisi de, alt seviyede oluşan bir istisnanın üst seviyelere taşınmasıdır. Örneğin, uzak veritabanına bağlantı sırasında hatanın oluşması farklı katmanlardan kaynaklanabilir. Karşı sunucuya ulaşılmasına rağmen veritabanı sunucusu kapalı olabilir, uzak sunucuya ağ problemleri dolayısıyla erişilemeyebilir veya erişim engellenebilir. Sonuç olarak yazılım için veritabanı bağlantısının yapılamadığı önemli olmakla birlikte, işlem sırasında oluşan istisnaların niteliklerinin değerlendirilmesi çözüme ulaşmada oldukça önemlidir.

Benzer kullanım, önerilen istisna modelinde de uygulanabilir. Örneğin veritabanına kullanıcının aynı anahtar değere sahip ikinci bir kaydı eklemeye çalışması durumunda SQLException istisnası oluşmasına rağmen (Veri katmanı tarafından) , bu istisna yakalanabilir, UserException olarak tekrar fırlatılabilir. Bu durumda InnerException olarak veri katmanından gelen SQLException istisna nesnesinin kullanımı mümkündür.

Modelde belirtilen teknik istisnalar için de aynı işlem uygulanabilir. Güvenlik kriterleri değerlendirildiğinde, oluşan bir teknik istisna hakkında verilen her bilginin kötü niyetli kişiler tarafından kullanılabileceği açıktır. TechnicalException isminde yeni bir istisna oluşturulabilir ve bu istisnanın InnerException özelliği gerçekte oluşan istisnayı belirtebilir.

Şekil 20, Exception.InnerException

Exception.InnerException kullanımı ile ilgili aşağıda örnek gösterilmiştir.

C#

**static** **class** Program {

// Donanımsal kontrollerin yapıldığı metottur. Bu metodun istisna   
 // fırlatmadan tamamlanması ağ donanımının başarıyla açıldığı anlamını   
 // taşımaktadır.

**static** **void** CheckHw() {

. . .

**throw** **new** HardwareException("Donanımsal Hata");

}

// Uzak sunucuya TCP bağlantısı yapan metottur. Metot öncelikle

// ağ donanımını kontrol etmektedir.

**static** **void** CheckNetwork() {

**try** {

CheckHw();

. . .

}

**catch** (Exception exc) {

// Ağ donanımında oluşan istisna exc ile ifadeedilmektedir. Metot,   
 // donanım hatasını belirtir istisnayı kapsayan ağ istisnası

// fırlatmaktadır.

**throw** **new** NetworkException("Ağ Hatası", exc);

}

}

// Veri tabanı bağlantısını gerçekleştiren metottur. Öncelikle uzak   
 // sunucuya TCP bağlantısı CheckNetwork metodu çağrılarak yapılmaktadır.

**static** **void** ConnectDb() {

**try** {

CheckNetwork();

. . .

}

**catch** (Exception exc) {

// Uzak sunucuya bağlanılamaması durumunda, ağ istisnasını kapsayan

// istisna fırlatılmaktadır.

**throw** **new** ConnectException("DBBağlantı Hatası", exc);

}

}

// Veri tabanına kayıt ekleyen metottur. Metot öncelikle veritabanı   
 // bağlantısı yapmaktadır.

**static** **void** Ekle(**string** ad, **string** soyad) {

**try** {

ConnectDb();

. . .

}

**catch** (Exception exc) {

// Oluşan istisna teknik bir istisna olarak değerlendirilmektedir.

**throw** **new** TechnicalException("Teknik Hata", exc);

}

}

// İstisna ve iç istisna detaylarını yazan yardımcı metottur. Oluşan istisna   
 // hakkında üretim ortamlarında kullanıcıya bu derecede detay içeren bilgi

// verilmesi önerilmemektedir. Buna rağmen istisnanın daha sonra

// incelenmek üzere tüm detayları ile saklanması problemin teşhisinde büyük   
 // önem arz etmektedir.

**static** **void** Yaz(Exception exc) {

Console.WriteLine("İstisna Tipi: {0}", exc.GetType().Name);

Console.WriteLine("Hata Mesajı : {0}", exc.Message);

Exception e = exc.InnerException;

**if** (e != **null**) {

Console.WriteLine("İç İstisna:");

Yaz(e);

e = e.InnerException;

}

}

**static** **void** Main(**string**[] args) {

**try** {

// Kullanıcı arabiriminden sadece Ekle metodu çağrılmaktadır. İstisna

// oluşması durumunda Exception.InnerException özelliği ile en derinde

// oluşan istisnaya kadar ulaşılabilmektedir.

Ekle("A", "B");

Console.WriteLine("Başarılı.");

}

**catch** (Exception exc) {

Yaz(exc);

}

}

}

VB.NET

**Class** Program

' Donanımsal kontrollerin yapıldığı metottur. Bu metodun istisna fırlatmadan   
 ' tamamlanması ağ donanımının başarıyla açıldığı anlamını taşımaktadır.

**Shared** **Sub** CheckHw()

. . .

**Throw** **New** HardwareException("Donanımsal Hata")

**End** **Sub**

' Uzak sunucuya TCP bağlantısı yapan metottur. Metot öncelikle

' ağ donanımını kontrol etmektedir.

**Shared** **Sub** CheckNetwork()

**Try**

CheckHw()

. . .

**Catch** exc **As** Exception

' Ağ donanımında oluşan istisna exc ile ifade edilmektedir. Metot, donanım

' hatasını belirtir istisnayı kapsayan ağ istisnası fırlatmaktadır.

**Throw** **New** NetworkException("Ağ Hatası", exc)

**End** **Try**

**End** **Sub**

' Veri tabanı bağlantısını gerçekleştiren metottur.

' Öncelikle uzak sunucuya TCP bağlantısı CheckNetwork metodu çağrılarak   
 ' yapılmaktadır.

**Shared** **Sub** ConnectDb()

**Try**

CheckNetwork()

. . .

**Catch** exc **As** Exception

' Uzak sunucuya bağlanılamaması durumunda, ağ istisnasını kapsayan

' istisna fırlatılmaktadır.

**Throw** **New** ConnectException("Bağlantı Hatası", exc)

**End** **Try**

**End** **Sub**

' Veri tabanına kayıt ekleyen metottur. Metot öncelikle veritabanı   
' bağlantısı yapmaktadır.

**Shared** **Sub** Ekle(**ByVal** ad **As** **String**, **ByVal** soyad **As** **String**)

**Try**

ConnectDb()

. . .

**Catch** exc **As** Exception

**Throw** **New** SystemException("Teknik Hata", exc)

**End** **Try**

**End** **Sub**

' İstisna ve iç istisna detaylarını yazan yardımcı metottur.

' Oluşan istisna hakkında üretim ortamlarında kullanıcıya bu derecede detay   
 ' içeren bilgi verilmesi önerilmemektedir. Buna rağmen istisnanın daha sonra

' incelenmek üzere tüm detayları ile saklanması problemin teşhisinde büyük   
 ' önem arz etmektedir.

**Shared** **Sub** Yaz(**ByVal** exc **As** Exception)

Console.WriteLine("İstisna Tipi: {0}", exc.**GetType**().Name)

Console.WriteLine("Hata Mesajı : {0}", exc.Message)

**Dim** e **As** Exception

e = exc.InnerException

**If** **Not** e **Is** **Nothing** **Then**

Console.WriteLine("İç İstisna:")

Yaz(e)

e = e.InnerException

**End** **If**

**End** **Sub**

**Shared** **Sub** Main(**ByVal** args() **As** **String**)

**Try**

' Kullanıcı arabiriminden sadece Ekle metodu çağrılmaktadır. İstisna

' oluşması durumunda Exception.InnerException özelliği ile en derinde

' oluşan istisnaya kadar ulaşılabilmektedir.

Ekle("A", "B")

Console.WriteLine("Başarılı.")

**Catch** exc **As** Exception

Yaz(exc)

**End** **Try**

**End** **Sub**

**End** **Class**

Kod 124, İç İstisna Nesnesi Kullanım Örnekleri

Kod çıktısı aşağıda gösterilmiştir.

İstisna Tipi: TechnicalException

Hata Mesajı : Teknik Hata

İç İstisna:

İstisna Tipi: NetworkException

Hata Mesajı : DBBağlantı Hatası

İç İstisna:

İstisna Tipi: NetworkException

Hata Mesajı : Ağ Hatası

İç İstisna:

İstisna Tipi: HardwareException

Hata Mesajı : Donanımsal Hata

**Exception.StackTrace Kullanımı**

StackTrace özelliği, istisna oluştuğunda çağrılan metotlar ve istisnanın oluştuğu kaynak kod dosya isimleri ile satır sayılarını belirtir değerdir. Problemlerin teşhisinde önemli kazanımlar sağlayabilen StackTrace özelliği ile ilgili örnek kullanım aşağıda gösterilmiştir.

| C# | VB.NET |
| --- | --- |
| **static** **void** A() {  **throw** **new** Exception("XYZ");  }  **static** **void** B() {  A();  }  **static** **void** Main(**string**[] args) {  **try** {  B();  }  **catch** (Exception exc) {  Console.WriteLine(exc.StackTrace);  }  } | **Private** **Shared** **Sub** A()  **Throw** **New** Exception("X")  **End** **Sub**  **Private** **Shared** **Sub** B()  A()  **End** **Sub**  **Shared** **Sub** Main(**ByVal** args() **As** **String**)  **Try**  B()  **Catch** exc **As** Exception  Console.WriteLine(exc.StackTrace)  **End** **Try**  **End** **Sub** |

Tablo 44, Exception.StackTrace Kullanım Örneği

Kod çıktısı aşağıda gösterilmiştir.

at ConsoleApplication.Program.A() in Module1.vb:line 11

at ConsoleApplication.Program.B() in Module1.vb:line 15

at ConsoleApplication4.Program.Main(String[] args) in   
 Module1.vb:line 20

**Exception.TargetSite Kullanımı**

Exception.TargetSite özelliği istisnanın oluştuğu metot hakkında daha detaylı bilgiye ulaşmayı sağlamaktadır. TargetSite özelliği System.Reflection.MethodBase sınıfı tipinde bir nesne döndürmektedir.

Aşağıda örnek kullanım gösterilmiştir.

C#

**static** **class** Program {

**static** **void** A() {

**throw** **new** Exception("XYZ");

}

**static** **void** Main(**string**[] args) {

**try** {

A();

}

**catch** (Exception exc) {

Console.WriteLine(exc.TargetSite.ToString());

Console.WriteLine(exc.TargetSite.DeclaringType.Name);

Console.WriteLine(exc.TargetSite.IsStatic);

}

}

}

VB.NET

**Class** Program

**Shared** **Sub** A()

**Throw** **New** Exception("XYZ")

**End** **Sub**

**Shared** **Sub** Main(**ByVal** args() **As** **String**)

**Try**

A()

**Catch** exc **As** Exception

Console.WriteLine(exc.TargetSite.ToString())

Console.WriteLine(exc.TargetSite.DeclaringType.Name)

Console.WriteLine(exc.TargetSite.IsStatic)

**End** **Try**

**End** **Sub**

**End** **Class**

Kod 125, Exception.TargetSite Kullanım Örnekleri

Kod çıktısı aşağıda gösterilmiştir.

Void A()

Program

True

Çıktının C# ve VB.NET için aynı olduğuna dikkat ediniz. Yansıma ile alınan metot bilgileri, hangi dille geliştirilmiş olursa olsun IL kod yapısında olduğu için çıktı aynı değerleri içermektedir.

Kullanıcı Tanımlı İstisna Sınıfları

Sınıf miras kurallarının geçerli olduğu kullanıcı tanımlı istisna sınıfları, System.Exception veya System.ApplicationException sınıfları miras alınarak yapılabilir. Daha önceki konularda aktarılan bilgiler çerçevesinde ata sınıf olarak ApplicationException sınıfı kullanımı önerilmektedir.

Aşağıda, istisna yönetim modeli içerisinde kullanılabilecek TechnicalException isimli sınıf gösterilmiştir.

C#

[**global**::System.Serializable]

**public** **class** TechnicalException : ApplicationException {

**public** TechnicalException() {

}

**public** TechnicalException(**string** message) : base(message) {

}

**public** TechnicalException(**string** message, Exception inner) :   
 **base**(message, inner) {

}

**protected** TechnicalException(

System.Runtime.Serialization.SerializationInfo info,

System.Runtime.Serialization.StreamingContext context)

: base(info, context) {

}

}

VB.NET

<System.Serializable()> \_

**Public** **Class** TechnicalException

**Inherits** Exception

**Public** **Sub** **New**()

**End** **Sub**

**Public** **Sub** **New**(**ByVal** message **As** **String**)

**MyBase**.**New**(message)

**End** **Sub**

**Public** **Sub** **New**(**ByVal** message **As** **String**, **ByVal** inner **As** Exception)

**MyBase**.**New**(message, inner)

**End** **Sub**

**Protected** **Sub** **New**(**ByVal** info **As** \_  
 System.Runtime.Serialization.SerializationInfo, \_  
 **ByVal** context **As** System.Runtime.Serialization.StreamingContext)

**MyBase**.**New**(info, context)

**End** **Sub**

**End** **Class**

Yukarıda gösterilen sınıf, yeni bir istisna sınıfı tanımlarken şablon olarak kullanılabilir. VS.NET kod şablonları yardımıyla da aynı işlem kolaylıkla yapılabilir. VS.NET kod şablonlarını kullanmak için editör üzerinde CTRL-J tuş takımını kullanıp Exception seçeneği üzerinde iken iki kez TAB tuşuna basmak yeterlidir.  
  
Serializable öz niteliği ve dördüncü yapıcı metodun amacı istisna nesnesinin serilizasyonunu sağlamaktır. Kitabın ilerleyen bölümlerinde nesne serilizasyonu hakkında detaylı bilgi sunulacaktır.

İstisna Yayınlama

İstisna yayınlama, uygulama yazılımında oluşan istisnaların daha sonra incelenmek üzere saklanması veya anında müdahale amacıyla e-posta, SMS gibi yöntemlerle ilgililere iletimini ifade eden kavramdır.

Oluşan istisnaların gerekli tüm detayları ile saklanması problemin teşhisinde büyük önem arz etmektedir. Sistemde oluşan bir hataya anında müdahale edebilmek maddiyat ve prestij kayıplarının en erken biçimde önüne geçebilmek için önemlidir.

Güvenlik boyutu ile değerlendirildiğinde, sisteme yapılan bir saldırı sonrası oluşan istisnaların incelenmesi, saldırı hakkında detay bilgiye ulaşmada ve yazılım zafiyetlerinin belirlenmesinde önemli rol oynayabilmektedir. Örneğin belirli bir zaman diliminde çok fazla hatalı kullanıcı adı ve şifre ile kimlik doğrulama (ing. Authentication) yapılmaya çalışılması ve sonucunda oluşan güvenlik istisnaları olası kötü niyetli kullanıcıyı tanımlamayı sağlayabilmektedir.

İstisna yayınlama işleminde özen gösterilmesi gereken konulardan birisi de hangi istisnaların yayınlanacağıdır. İstisnaların birbirlerinden ayırt edilebilmesi için farklı istisna sınıflarının tanımlanması ve farklı katmanlardan iletilen istisnaların süzülerek kullanıcı ve teknik istisna sınıflarına dönüşümü kritiktir. İstisna filtreleme (ing. Exception Filtering) olarak tanımlanabilecek bu işlevin yapılabilmesi için yazılımlarda sağlam istisna yönetim modelinin kurulması zorunluluktur.

Sistemde oluşan tüm istisnaların herhangi bir filtrelemeden geçmeden yayınlanması, inceleme süresini artıracağı gibi önemli istisnaların da göz ardı edilebilme riskini beraberinde getirecektir. Temel prensip olarak, sadece teknik ve güvenlik istisnalarının yayınlanması yeterli gözükmektedir.

Aynı istisnanın birden fazla yayınlanmaması için yönetimin tek noktadan yapılması önerilmektedir. Son olarak, istisnanın yayınlanması sırasında oluşabilecek istisnaların yönetiminin de yapılıyor olması sağlık bir istisna yönetimi için dikkat edilmesi gereken konulardandır.

Bir sonraki konuda, önerilen istisna yönetim modeli çerçevesinde, istisna yayınlama ile ilgili bilgiler sunulacaktır.

İstisna Yönetim Modeli Gerçekleştirimi

Bu konu başlığında, daha önce çerçevesi çizilen istisna yönetim modeli hakkında daha detaylı açıklamalar yapılacak ve örneklemeler sunulacaktır.

Tip Tanımları

Model gerçekleştirimi için aşağıda listelenen tipler kullanılacaktır. Modelin tüm uygulama yazılımlarında kullanılabilirliği de değerlendirerek gerektiğinde genişletilebilmesini sağlamak amacıyla bazı işlevler arabirimler yardımıyla sağlanacaktır.

| Tip | Açıklama |
| --- | --- |
| IExceptionPublisher | Sistemde oluşan istisnaların yayınlanabilmesini standart bir yapı ile sağlamak amacıyla tanımlanan arabirimdir. |
| IExceptionFilter | İstisnaların yayınlanmadan önce filtrelenebilmesini sağlamak amacıyla kullanılan arabirimdir. |
| UserException | Kullanıcı hatasından kaynaklı istisnalar için ata sınıftır. |
| ApplicationSecurityException | Uygulamada oluşan güvenlik istisnaları için kullanılan sınıftır. Bu sınıf UserException sınıfını miras almaktadır. |
| TechnicalException | Kullanıcı hatasından kaynaklanmayan, teknik problemlere işaret eden istisnalar için ata sınıftır. |
| ExceptionManager | İstisna yönetiminin farklı metotları yardımıyla yapılabileceği temel istisna yönetim sınıfıdır. |

Tablo 45, Önerilen İstisna Modeli Tipleri

UserException ve TechnicalException sınıfları System.ApplicationException sınıfını miras almaktadır.

Aşağıda IExceptionPublisher, IExceptionFilter arabirimleri ve ExceptionManager sınıf üyeleri açıklamaları ile birlikte gösterilmiştir.

C#

**public** **interface** IExceptionPublisher {

// exc ile iletilen istisna nesnesinin yayınlanmasını sağlayan metot tanımı.

**void** Publish(Exception exc);

}

**public** **interface** IExceptionFilter {

// exc ile belirtilen istisnanın yayınlanıp yayınlanmayacağını

// belirten fonksiyon tanımı. Fonksiyonun True değerini döndürmesi   
 // istisnanın yayınlanması gerektiğini, False değerini döndürmesi

// yayınlanmaması gerektiğini belirtmektedir.

**bool** Filter(Exception exc);

}

**public** **sealed** **class** ExceptionManager {

// İstisnanın tek noktadan yayınlanmasını sağlayan yardımcı metot. Uygulama

// içerisinden bu metodun çağrımı istisnanın yayımlanması için

// yeterli olacaktır.

**public** **static** **void** Publish(Exception exc) {

}

// exc ile iletilen herhangi istisna sınıfını kontrol eden ve

// istisna tipine göre sadece UserException veya TechnicalException döndüren

// yardımcı metottur. Bu metot yardımıyla tek noktadan istisna nesnelerinin   
 // dönüşümü mümkündür.

**public** **static** ApplicationException Convert(Exception exc) {

}

}

VB.NET

**Public** **Interface** IExceptionPublisher

' exc ile iletilen istisna nesnesinin yayınlanmasını sağlayan metot tanımı.

**Sub** Publish(**ByVal** exc **As** Exception)

**End** **Interface**

**Public** **Interface** IExceptionFilter

' exc ile belirtilen istisnanın yayınlanıp yayınlanmayacağını belirten   
 ' fonksiyon tanımı. Fonksiyonun True değerini döndürmesi istisnanın

' yayınlanması gerektiğini, False değerini döndürmesi yayınlanmaması   
 ' gerektiğini belirtmektedir.

**Function** Filter(**ByVal** exc **As** Exception)

**As** **Boolean**

**End** **Interface**

**Public** **NotInheritable** **Class** ExceptionManager

' İstisnanın tek noktadan yayınlanmasını sağlayan yardımcı metot. Uygulama

' içerisinden bu metodun çağrımı istisnanın yayımlanması için

' yeterli olacaktır.

**Public** **Shared** **Sub** Publish(**ByVal** exc **As** Exception)

**End** **Sub**

' exc ile iletilen herhangi istisna sınıfını kontrol eden ve

' istisna tipine göre sadece UserException veya TechnicalException döndüren

' yardımcı metottur. Bu metot yardımıyla tek noktadan istisna nesnelerinin   
 ' dönüşümü mümkündür.

**Public** **Shared** **Function** Convert(**ByVal** exc **As** Exception) **As** AplicationException

**End** **Function**

**End** **Class**

Tüm tiplerin gerçekleştirimi VS.NET sınıf kütüphanesi projesi içerisinde yapılmış ve tek Assembly içerisinde toplanmıştır.

Gerçekleştirim Detayları

Model gerçekleştiriminin daha iyi anlaşılabilmesi için önemli görülen bazı sınıflar ve metotlar hakkında bilgiler verilecektir.

İstisna Yönetici Sınıfı

ExceptionManager sınıfı istisnanın yayınlanmasını ve dönüşümünü sağlayan iki adet statik yardımcı sınıf içermektedir.

Statik yapıcı metot içerisinde, programcının konfigürasyon sistemi ile belirttiği IExceptionPublisher ve IExceptionFilter arabirim gerçekleştirimlerini yapan nesneler oluşturulmaktadır. Bu nesneler Publish metodu içerisinde kullanılmaktadır.

C#

**public** **sealed** **class** ExceptionManager {

// Alanların IExceptionPublisher ve IExceptionFilter arabirimi tipinde   
 // tanımlandığına dikkat ediniz.

**private** **static** IExceptionPublisher publisher;

**private** **static** IExceptionFilter filter;

**static** ExceptionManager() {

// Activator ve Type sınıfları kullanılarak publisher ve filter nesneleri

// oluşturulmaktadır.

}

**public** **static** **void** Publish(Exception exc) {

// İstisna filtreleyicisi tarafından istisnanın yayınlanması gerektiğinin

// belirtilmesi durumunda istisna yayınlayıcı nesnesi

// Publish metodu çağrılmaktadır.

**if** (filter.Filter(exc))

publisher.Publish(exc);

}

**private** ExceptionManager() {

// ExceptionManager tipinde bir nesne oluşturmanın önüne

// geçmek amacıyla yapıcı metot private olarak tanımlanmıştır.

}

. . .

VB.NET

**Public** **NotInheritable** **Class** ExceptionManager

' Alanların IExceptionPublisher ve IExceptionFilter arabirimi tipinde   
 ' tanımlandığına dikkat ediniz.

**Private** **Shared** publisher **As** IExceptionPublisher

**Private** **Shared** filter **As** IExceptionFilter

**Shared** **Sub** **New**()

' Activator ve Type sınıfları kullanılarak publisher ve

' filter nesneleri oluşturulmaktadır.

**End** **Sub**

**Public** **Shared** **Sub** Publish(**ByVal** exc **As** Exception)

' İstisna filtreleyicisi tarafından istisnanın yayınlanması gerektiğinin

' belirtilmesi durumunda istisna yayınlayıcı nesnesi Publish metodu   
 ' çağrılmaktadır.

**If** filter.Filter(exc) **Then**

publisher.Publish(exc)

**End** **If**

**End** **Sub**

**Private** **Sub** **New**()

' ExceptionManager tipinde bir nesne oluşturmanın önüne geçmek amacıyla   
 ' yapıcı metot private olarak tanımlanmıştır.

**End** **Sub**

. . .

ExceptionManager sınıfının miras alınamaz ve sınıf tipinde nesne oluşturulamaz şekilde tanımlandığına dikkat ediniz. Sistemde oluşan istisnaların yönetimi güvenlik açısından oldukça önemlidir. Kötü niyetli bir kullanıcının bu sınıfı miras alabilmesi ve istisna yöntemi mantığını değiştirebilmesi riski böylece engellenmiştir.

ExceptionManager.Convert metodu, farklı tipte olan istisnaların ApplicationException tipine dönüşümünü sağlayan metottur. Metot sadece UserException veya TechnicalException nesneleri döndürmektedir. Böylece filtreleme sırasında net olarak oluşan istisnanın niteliği anlaşılabilmektedir.

C#

**public** **static** ApplicationException Convert(Exception exc) {

// UserException tipinde ise aynen döndür.

**if** (exc **is** UserException)

**return** (UserException)exc;

// TechnicalException tipinde ise aynen döndür.

**else** **if** (exc **is** TechnicalException)

**return** (TechnicalException)exc;

// Her iki istisna tipinde değilse teknik hata olarak döndür ve   
 // parametre olarak iletilen istisna nesnesini yapıcı metoda ilet.

**else**

**return** **new** TechnicalException("Sistemde teknik hata oluştu.", exc);

}

VB.NET

**Public** **Shared** **Function** Convert(**ByVal** exc **As** Exception)**As** ApplicationException

' UserException tipinde ise aynen döndür.

**If** **TypeOf** exc **Is** UserException **Then**

**Return** **CType**(exc, UserException)

' TechnicalException tipinde ise aynen döndür.

**ElseIf** **TypeOf** exc **Is** TechnicalException **Then**

**Return** **CType**(exc, TechnicalException)

' Her iki istisna tipinde değilse teknik hata olarak döndür ve   
 ' parametre olarak iletilen istisna nesnesini yapıcı metoda ilet.

**Else**

**Return** **New** TechnicalException("Sistemde teknik hata oluştu.", exc)

**End** **If**

**Function**

İstisna Yayınlayıcı ve Filtre Sınıfları

Modelin arabirimler aracılığı ile çalışma yöntemini benimsemesi önemli bir avantajı da beraberinde getirmektedir. Programcı, IExceptionPublisher ve IExceptionFilter arabirim gerçekleştirimlerini yapan sınıfları geliştirebilmekte, böylece model yapısı çerçevesinde istediği şekilde istisna yayınlama ve filtreleme işlevini sağlayabilmektedir.

ExceptionManager sınıfı, statik yapıcı metot içerisinde programcı tarafından tanımlanan nesneleri oluşturmakta ve nesne gerçekleştirim detayları ile ilgilenmeden sadece arabirim metotlarını kullanmaktadır.

C#

**public** **sealed** **class** ExceptionManager {

**private** **static** IExceptionPublisher publisher;

**private** **static** IExceptionFilter filter;

**static** ExceptionManager() {

Type type;

// Konfigürasyon sistemi yardımıyla sınıf tanımı ve Assembly adı  
 // aracılığı ile sınıf tip bilgisini al.

type = Type.GetType(  
 ConfigurationManager.AppSettings["ExceptionPublisher"], **true**);  
  
 // Alınan tip bilgisine göre nesneyi oluştur.

publisher = (IExceptionPublisher)Activator.CreateInstance(type);

type = Type.GetType(  
 ConfigurationManager.AppSettings["ExceptionFilter"], **true**);

filter = (IExceptionFilter)Activator.CreateInstance(type);

}

. . .

VB.NET

**Public NotInheritable Class** ExceptionManager

**Private Shared** publisher **As** IExceptionPublisher

**Private Shared** filter **As** IExceptionFilter

**Shared** **Sub** **New**()

**Dim** type **As** Type

' Konfigürasyon sistemi yardımıyla sınıf tanımı ve Assembly adı  
 ' aracılığı ile sınıf tip bilgisini al.

type = type.**GetType**(  
 ConfigurationManager.AppSettings("ExceptionPublisher"), **True**)

' Alınan tip bilgisine göre nesneyi oluştur.

publisher = **CType**(Activator.CreateInstance(type), IExceptionPublisher)

type = type.**GetType**(  
 ConfigurationManager.AppSettings("ExceptionFilter"), **True**)

filter = **CType**(Activator.CreateInstance(type), IExceptionFilter)

**End** **Sub**

. . .

Konfigürasyon dosyası Windows uygulamaları için app.config, web uygulamaları için web.config isimli dosyalardır. Farklı konfigürasyon sistemlerinin uygulamalarda kullanılabilmesine rağmen örnek modelde kullanım kolaylığı sebebi ile dosya bazlı konfigürasyon yöntemi seçilmiştir.   
  
Örnek konfigürasyon dosya içeriği aşağıda gösterilmiştir.  
  
 <appSettings>  
 <add key="ExceptionPublisher" value="SamplePublisher, Exceptions" />

<add key="ExceptionFilter" value="ExceptionFilter, Filters" />  
</appSettings>

Kullanım Örneği

Aşağıda örnek Windows uygulamasında modelin kullanımı gösterilmiştir.

C#

// Yardımcı metot.

**private** **void** ManageException(Exception exc) {

// exc parametresi ile iletilen istisna nesnesiExceptionManager.Convert

// yardımcı metodu ile ApplicationException tipine dönüştürülmektedir.

ApplicationException appExc = ExceptionManager.Convert(exc);

// İstisna yayınlanmaktadır. Bu metodun çağrımı istisnanın

// mutlaka yayınlanacağı anlamını taşımamaktadır. Tanımlı istisna

// filtreleyici sınıf gerekli yönetimi yapmaktadır.

ExceptionManager.Publish(appExc);

// İstisna mesajı kullanıcıya gösterilmektedir.

// Dikkat edilmesi gereken konu, aşağıdaki örnekte NullReferenceException   
 // istisna nesnesi fırlatılmış olmasına rağmen gösterilen appExc ile

// ifade edilen istisnanın TechnicalException sınıfına ait olduğudur.

MessageBox.Show(appExc.Message);

}

// Örnek Kullanım:

**try** {

// Henüz oluşturulmamış bir nesne üye girişi sonucu istisna oluşacaktır.

ArrayList list = **null**;

list.Add("");

}

**catch** (Exception exc) {

// Yakalanan istisnanın yönetimi yapılmaktadır.

ManageException(exc);

}

VB.NET

// Yardımcı metot.

**Private** **Sub** ManageException(**ByVal** exc **As** Exception)

' exc parametresi ile iletilen istisna nesnesi ExceptionManager.Convert

' yardımcı metodu ile ApplicationException tipine dönüştürülmektedir.

**Dim** appExc **As** ApplicationException = ExceptionManager.Convert(exc)

' İstisna yayınlanmaktadır. Bu metodun çağrımı istisnanın mutlaka   
 ' yayınlanacağı anlamını taşımamaktadır. Tanımlı istisna filtreleyici sınıf   
 ' gerekli yönetimi yapmaktadır.

ExceptionManager.Publish(appExc)

' İstisna mesajı kullanıcıya gösterilmektedir. Dikkat edilmesi gereken konu,

' aşağıdaki örnekte NullReferenceException istisna nesnesi fırlatılmış   
 ' olmasına rağmen gösterilen appExc ile ifade edilen istisnanın

' TechnicalException sınıfına ait olduğudur.

MessageBox.Show(appExc.Message)

**End** **Sub**

' Örnek Kullanım:

**Try**

' Henüz oluşturulmamış bir nesne üye girişi sonucu istisna oluşacaktır.

**Dim** list **As** ArrayList = **Nothing**

list.Add("")

**Catch** exc **As** Exception

' Yakalanan istisnanın yönetimi yapılmaktadır.

ManageException(exc)

**End** **Try**

Listelenen kaynak kodlar \Bölüm 7\Model klasöründe bulunmaktadır.

İnsana, öğrenmeyi ancak kendi içinde bulacağı dışında hiçbir şey öğretemezsiniz.

Galileo

Bölüm

8

Delegeler ve Olaylar

K

itabın daha önceki konularında, nesneler arası haberleşmenin standart yöntemlerle yapılabilmesi için arabirim (ing. Interface) kavramı ve arabirimlerin kullanımı hakkında bilgiler verilmişti.

Bu bölümde, .NET içerisinde nesnelerin birbirleri ile haberleşmeleri ve birbirlerini olaylardan haberdar edebilmeleri hakkında bilgiler sunulacaktır.

Nesne Haberleşmesi

Tekrar kullanılabilir, bir başka anlatımla bir kez geliştirilip sınıf kütüphanesi olarak derlenen ve her yazılım projesinde kullanılabilen, sınıflar programcılara büyük kazanımlar sağlamaktadır. Tekrar kullanılabilirliğin en etkin uygulanabilmesi için gerekli şartlardan birisi de, birbirleri hakkında bilgisi olmayan nesnelerin standart yöntemlerle haberleşmelerini sağlayabilmektir.

Haberleşmenin standart yöntemler ile sağlanmasında arabirimler önemli rol oynamaktadır. Arabirimler aracılığı ile haberleşme ve arabirim kullanım detayları kitabın daha önceki konularında aktarılmıştı.

Arabirimler ile haberleşmeye alternatif olan delegeler ve olaylar, nesne haberleşmesinin daha üst seviye ve daha az kod geliştirerek yapılabilmesini sağlamaktadır.

Nesne haberleşmesini sağlamak amacıyla gerek arabirimler gerek delege ve olaylar kullanılabilir. Hangi yöntemin ne zaman kullanılması gerektiği konusu bu bölümün tamamlanması ile netlik kazanacaktır.

Örnek Senaryo

Konuyu daha iyi aktarabilmek için anlatım aşağıdaki senaryo bazında yapılacaktır.

ABC firması, .NET ile yazılım geliştiren programcıların kullanabileceği ve bilgisayarın USB portu ile haberleşebilmeyi sağlayan bir kütüphane geliştirmek istemektedir. Firmanın amacı geliştirdiği sınıfı bir Assembly olarak uygun lisans ücretleri ile dağıtmaktır.  
  
Konuyla ilgili programcıların fikirlerini alan ABC firması, sınıf yardımıyla USB portuna bağlanıldıktan sonra, bilgisayar ile porta bağlı cihaz arasında gidip gelen verilerin saklanabilmesi veya kullanıcıya gösterilebilmesinin kolaylıkla yapılabilmesinin istendiğini değerlendirmiştir.  
  
Programcıların önerilerini dikkate alan ABC firması, USB isminde bir sınıf ve bu sınıfın çeşitli metotlarını tanımlamıştır. Programcıların USB nesnesine gerekli bildirimleri yapması ardından, nesnenin o anki durumu, porta gönderilen ve porttan alınan verilerin programcı tarafından tanımlanan nesnelere iletimi delege ve olay kullanımı ile kolaylıkla yapılmıştır.

Delegeler

Delegeler (ing. Delegates), CTS içerisinde System.Delegate sınıfı ile tanımlı tiplerdir. Bir delege tanımı ile yapılan işlem, sınıf veya arabirim tanımı benzeri, yeni bir tip tanımıdır.

Delege Tanımı

Rezerve bir kelime olan delegate ile yapılan delege tanımı aşağıda gösterilmiştir.

C#

**delegate** **void** Hata(**string** mesaj);

VB.NET

**Delegate** **Sub** Hata(**ByVal** mesaj **As** **String**)

Kod 126, Örnek Delege Tanımı

Delege kullanımı ile hedeflenen, parametre listesi ve döndürdüğü değer belirli olan bir metodu tanımlamaktır. Bu yüzden delegeler fonksiyon işaretçileri (ing. Function Pointers) olarak da anılmaktadır.

Delege tanımı;

* Delegenin adı
* Belirttiği metodun döndürdüğü değer
* Metot parametre listesinden

Oluşmaktadır.

Yukarıdaki örnekte Hata isminde bir delege tanımı yapılmıştır. Tanımlanan delege, değer döndürmeyen ve string tipinde parametre olan bir metodu belirtmektedir.

Delege isimlendirmelerinde, delege isminin EventHandler ile bitmesi bir zorunluluk olmamasına rağmen önerilen isimlendirme standardıdır.

Delege kavramının daha iyi anlaşılabilmesi için aşağıda temel delege kullanım yöntemini gösteren örneği inceleyiniz.

C#

**class** DosyaLogger {

**public** **static** **void** LogYaz(**string** s) {

}

}

**class** EkranLogger {

**public** **static** **void** Goster(**string** s) {

}

}

**static** **class** Program {

// LogEventHandler isminde, değer döndürmeyen, bir adet string tipinde   
 // parametre alan metodu belirten delege tanımı yapılmıştır.

**public** **delegate** **void** LogEventHandler(**string** msg);

**static** **void** Main(**string**[] args) {

// d1 isminde delege nesnesi tanımlanmıştır.

LogEventHandler d1;

// d1 isimli değişkenin DosyaLogger sınıfının LogYaz metoduna işaret etmesi

// sağlanmıştır.

d1 = **new** LogEventHandler(DosyaLogger.LogYaz);

LogEventHandler d2;

// Aynı işlem aşağıdaki şekilde de yapılabilir.

d2 = EkranLogger.LogGoster;

// Aşağıdaki kodlar ile yapılan işlem sırasıyla DosyaLogger.LogYaz ve

// EkranLogger.LogGoster metotlarının çağrımıdır.

d1("Dosya");

d2("Ekran");

}

}

VB.NET

**Class** DosyaLogger

**Public** **Shared** **Sub** LogYaz(**ByVal** s **As** **String**)

**End** **Sub**

**End** **Class**

**Class** EkranLogger

**Public** **Shared** **Sub** Goster(**ByVal** s **As** **String**)

**End** **Sub**

**End** **Class**

**Class** Program

' LogEventHandler isminde, değer döndürmeyen, bir adet string tipinde   
 ' parametre alan metodu belirten delege tanımı yapılmıştır.

**Public** **Delegate** **Sub** LogEventHandler(**ByVal** msg **As** **String**)

**Shared** **Sub** Main(**ByVal** args() **As** **String**)

' d1 isminde delege nesnesi tanımlanmıştır.

**Dim** d1 **As** LogEventHandler

' d1 isimli değişkenin DosyaLogger sınıfının LogYaz metoduna işaret etmesi

' sağlanmıştır.

d1 = **New** LogEventHandler(**AddressOf** DosyaLogger.LogYaz)

**Dim** d2 **As** LogEventHandler

' Aynı işlem aşağıdaki şekilde de yapılabilir.

d2 = **AddressOf** EkranLogger.Goster

' Aşağıdaki kodlar ile yapılan işlem sırasıyla DosyaLogger.LogYaz ve

' EkranLogger.LogGoster metotlarının çağrımıdır.

d1("Dosya")

d2("Ekran")

**End** **Sub**

**End** **Class**

Kod 127, Basit Delege Kullanım Örneği

Nasıl ki bir tamsayı tipi ile hedeflenen içerisinde tamsayıları saklayabilecek bir tip tanımı ise, delege tanımı ile hedeflenen döndürdüğü değer ve parametre listesi belirli bir metot adresini tutabilmektir.

Örnekten de görüleceği üzere, delege değişkenine atanan metodun adı ve parametre isimleri önemli değildir. Önemli olan metodun döndürdüğü değer, parametre sayısı ve tipleridir.

delegate kelimesi ile derleyici tarafından arka planda yapılan işlem System.MultiCastDelegate sınıfını miras alan yeni bir sınıf tanımı yapmaktır.

Delege Kullanımı

Delegeler birden fazla metot adresini saklayabilmektedir. Böylece, bir delege çağrımı ile arka planda, delege tarafından saklı tüm metotlar eklenme sırasıyla otomatik olarak çağrılmaktadır.

C#

**class** USBPort {

// USB Portun durumunu belirten sayılabilir tip tanımı aşağıda yapılmıştır.

**public** **enum** UsbDurum {

Açık,

Kapalı

}

// Aktif USB nesnesi port durumunu belirten alan tanımı.

**private** UsbDurum durum = UsbDurum.Kapalı;

// Nesne tarafından USB porta bağlanılması veya port bağlantısının   
 // kapatılması durumunda haberleşme amacıyla kullanılan delege tanımı aşağıda

// yapılmıştır.

**public** **delegate** **void** UsbDurumEventHandler(UsbDurum status);

// Delege tipinde alan tanımı.

**private** UsbDurumEventHandler durumDelegesi = **null**;

**public** **void** Aç() {

// Programcı bu metodu çağırarak nesnenin USB porta bağlanmasını  
 // sağlamaktadır.

durum = UsbDurum.Açık;

// durumDelegesi isimli alana geçerli bir değer atanmış ise bu delegeyi  
 // çağır.

**if** (durumDelegesi != **null**)

durumDelegesi(UsbDurum.Açık);

}

**public** **void** Kapat() {

durum = UsbDurum.Kapalı;

**if** (durumDelegesi != **null**)

durumDelegesi(UsbDurum.Kapalı);

}

// Metodun kullanım amacı, nesnenin durumu değiştiğinde haberdar olabilmek   
 // amacıyla programcının istediği metodun kaydını yaptırabilmesidir.

**public** **void** DurumEventHandlerEkle(UsbDurumEventHandler handler) {

// handler isimli parametre ile belirtilen metot adresinidelege içerisine   
 // ekle.

durumDelegesi += handler;

}

// Metodun kullanım amacı, daha önce delegeye eklenen bir metodun  
 // çıkarılmasıdır.

**public** **void** DurumEventHandlerÇıkar(UsbDurumEventHandler handler) {

// handler isimli parametre ile belirtilen metot adresini delege   
 // içerisinden çıkar.

durumDelegesi -= handler;

}

}

**static** **class** Program {

// USBPort nesnesinin durumu değiştiğinde yeni durumu kullanıcıya göstermek   
 // amacıyla kullanılan metottur.

**static** **void** DurumGoster(USBPort. UsbDurum durum) {

Console.WriteLine("USB Port Durumu Değişti");

Console.WriteLine("Yeni Durum: {0}", durum);

}

// USBPort nesnesinin durumu değiştiğinde yeni durumu kaydetmek amacıyla

// kullanılan metottur.

**static** **void** DurumLogla(USBPort. UsbDurum durum) {

Console.WriteLine("Yeni durum kaydedildi");

Console.WriteLine("Kaydedilen durum: {0}", durum);

}

**static** **void** Main(**string**[] args) {

// USBPort nesnesi oluştur.

USBPort port = **new** USBPort();

// DurumGoster isimli metodu nesneye ilet.

port.DurumEventHandlerEkle(DurumGoster);

// DurumLogla isimli metodu nesneye ilet.

port.DurumEventHandlerEkle(DurumLogla);

// USB porta bağlanmak için Aç isimli metodu çağır.

port.Aç();

port.Kapat();

}

}

VB.NET

**Class** USBPort

' USB Portun durumunu belirten sayılabilir tip tanımı aşağıda

' yapılmıştır.

**Public** **Enum** UsbDurum

Açık

Kapalı

**End** **Enum**

' Aktif USB nesnesi port durumunu belirten alan tanımı.

**Private** durum **As** UsbDurum = UsbDurum.Kapalı

' Nesne tarafından USB porta bağlanılması veya port bağlantısının   
 ' kapatılması durumunda haberleşme amacıyla kullanılan delege tanımı aşağıda

' yapılmıştır.

**Public** **Delegate** **Sub** UsbDurumEventHandler(**ByVal** status **As** UsbDurum)

' Delege tipinde alan tanımı.

**Private** durumDelegesi **As** UsbDurumEventHandler = **Nothing**

' Programcı bu metodu çağırarak nesnenin USB porta bağlanmasını  
 ' sağlamaktadır.

**Public** **Sub** Aç()

durum = UsbDurum.Açık

' durumDelegesi isimli alana geçerli bir değer atanmış ise bu delegeyi   
 ' çağır.

**If** **Not** durumDelegesi **Is** **Nothing** **Then**

durumDelegesi(UsbDurum.Açık)

**End** **If**

**End** **Sub**

**Public** **Sub** Kapat()

durum = UsbDurum.Kapalı

**If** **Not** durumDelegesi **Is** **Nothing** **Then**

durumDelegesi(UsbDurum.Kapalı)

**End** **If**

**End** **Sub**

' Metodun kullanım amacı, nesnenin durumu değiştiğinde haberdar olabilmek   
 ' amacıyla programcının istediği metodun kaydını yaptırabilmesidir.

**Public** **Sub** DurumEventHandlerEkle(**ByVal** handler **As** sbDurumEventHandler)

' handler isimli parametre ile belirtilen metot adresini delege içerisine   
 ' ekle.

**Dim** temp **As** System.**Delegate**

temp = System.**Delegate**.Combine(durumDelegesi, handler)

durumDelegesi = **CType**(temp, UsbDurumEventHandler)

**End** **Sub**

' Metodun kullanım amacı, daha önce delegeye eklenen bir metodun   
 ' çıkarılmasıdır.

**Public** **Sub** DurumEventHandlerÇıkar(**ByVal** handler **As** UsbDurumEventHandler)

' handler isimli parametre ile belirtilen metot adresini delege içerisinden   
 ' çıkar.

**Dim** temp **As** System.**Delegate**

temp = System.**Delegate**.Remove(durumDelegesi, handler)

durumDelegesi = **CType**(temp, UsbDurumEventHandler)

**End** **Sub**

**End** **Class**

**Class** Program

' USBPort nesnesinin durumu değiştiğinde yeni durumu

' kullanıcıya göstermek amacıyla kullanılan metottur.

**Shared** **Sub** DurumGoster(**ByVal** durum **As** USBPort.UsbDurum)

Console.WriteLine("USB Port Durumu Değişti")

Console.WriteLine("Yeni Durum: {0}", durum)

**End** **Sub**

' USBPort nesnesinin durumu değiştiğinde yeni durumu kaydetmek amacıyla

' kullanılan metottur.

**Shared** **Sub** DurumLogla(**ByVal** durum **As** USBPort.UsbDurum)

Console.WriteLine("Yeni durum kaydedildi")

Console.WriteLine("Kaydedilen durum: {0}", durum)

**End** **Sub**

**Shared** **Sub** Main(**ByVal** args() **As** **String**)

' USBPort nesnesi oluştur.

**Dim** port **As** USBPort = **New** USBPort()

' DurumGoster isimli metodu nesneye ilet.

port.DurumEventHandlerEkle(**AddressOf** DurumGoster)

' DurumLogla isimli metodu nesneye ilet.

port.DurumEventHandlerEkle(**AddressOf** DurumLogla)

' USB porta bağlanmak için Aç isimli metodu çağır.

port.Aç()

port.Kapat()

**End** **Sub**

**End** **Class**

Kod 128, Delege Kullanımı

Örnek kod çıktısı aşağıda gösterilmiştir.

USB Port Durumu Değişti

Yeni Durum: Açık

Yeni durum kaydedildi  
Kaydedilen durum: Açık

USB Port Durumu Değişti

Yeni Durum: Kapalı

Yeni durum kaydedildi

Kaydedilen durum: Kapalı

Örnekte kullanılan durumDelegesi isimli alan içerisinde DurumGoster ve DurumLogla metotları saklanmaktadır. Böylece bu delegenin kullanımı ile her iki metot sırayla otomatik olarak çağrılmaktadır.

Listelenen kaynak kodlar \Bölüm 8\Delegates klasöründe bulunmaktadır.

Delege Parametre Listesi

Delegelerin belirttiği metotların parametreleri için, metot parametre kurallarına uyan her türlü parametre niteleyiciyi kullanmak mümkündür. Böylece, örneğin ref veya ByRef (sırasıyla C# ve VB.NET) parametre niteleyicileri kullanarak parametre değerlerinin değiştirilmesi mümkündür.

C#

**class** TestClass {

**public** **delegate** **void** TestDelegate(**ref** **int** i);

**public** TestDelegate delege;

**public** **void** TestMethod() {

**int** j = 12;

Console.WriteLine("Varsayılan Değer: {0}", j);

// Çağrım yapılmaktadır.

delege(**ref** j);

Console.WriteLine("Geri alınan Değer: {0}", j);

}

}

**static** **class** Program {

**static** **void** Değiştir(**ref** **int** i) {

i = 5;

}

**static** **void** Main(**string**[] args) {

TestClass t = **new** TestClass();

t.delege = Değiştir;

t.TestMethod();

}

}

VB.NET

**Class** TestClass

**Public** **Delegate** **Sub** TestDelegate(**ByRef** i **As** **Integer**)

**Public** delege **As** TestDelegate

**Public** **Sub** TestMethod()

**Dim** j **As** **Integer** = 12

Console.WriteLine("Varsayılan Değer: {0}", j)

' Çağrım yapılmaktadır.

delege(j)

Console.WriteLine("Geri alınan Değer: {0}", j)

**End** **Sub**

**End** **Class**

**Class** Program

**Shared** **Sub** Değiştir(**ByRef** i **As** **Integer**)

i = 5

**End** **Sub**

**Shared** **Sub** Main(**ByVal** args() **As** **String**)

**Dim** t **As** TestClass = **New** TestClass()

t.delege = **AddressOf** Değiştir

t.TestMethod()

**End** **Sub**

**End** **Class**

Kod 129, Parametre Niteleticileri ile Delege Kullanımı

Kod çıktısı aşağıda gösterilmiştir.

Varsayılan Değer: 12

Geri alınan Değer: 5

Parametre niteleyicileri ile nesneler arası bilgi alışverişi sağlanmaktadır. Böylece delegeler ile, temel amaç olan haberdar etme işlevine ek olarak, nesne davranışlarını da özelleştirebilme işlevi yapılabilmektedir.

Parametre niteleyicileri konusunda kitabın daha önceki bölümlerinde önemli bilgiler verilmiştir. Niteleyicilerin çalışma yönteminin değer ve referans tipler için aynı olmasına rağmen elde edilen sonuçların farklı olacağı unutulmamalıdır.

Önerilen Delege Parametre Listesi Kullanımı

USBPort sınıfı örneğinde, nesnenin USB porta bağlanması ve bağlantının kesilmesi sırasında delegeler yardımıyla otomatik olarak DurumGoster ve DurumLogla isimli metotlar çağrılmaktadır. Metotlara nesnenin durumunu belirtir değer iletilmekte, böylece programcılar USBPort nesnesinin durumu değiştiğinde gereken işlemleri yapabilmektedir.

C#

USBPort port1 = **new** USBPort();

USBPort port2 = **new** USBPort();

port1.DurumEventHandlerEkle(DurumGoster);

port1.DurumEventHandlerEkle(DurumLogla);

port2.DurumEventHandlerEkle(DurumGoster);

port2.DurumEventHandlerEkle(DurumLogla);

port1.Aç()

port2.Aç()

VB.NET

**Dim** port1 **As** USBPort = **New** USBPort()

**Dim** port2 **As** USBPort = **New** USBPort()

port1.DurumEventHandlerEkle(**AddressOf** DurumGoster)

port1.DurumEventHandlerEkle(**AddressOf** DurumLogla)

port2.DurumEventHandlerEkle(**AddressOf** DurumGoster)

port2.DurumEventHandlerEkle(**AddressOf** DurumLogla)

port1.Aç()

port2.Aç()

Yukarıdaki örnekte iki adet USBPort nesnesi oluşturulmuş ve her iki nesneye de aynı DurumGoster ve DurumLogla isimli metotlar iletilmiştir.

Geçerli bir kullanım olmasına rağmen, DurumGoster ve DurumLogla isimli metotlar içerisinde çağrımın hangi nesne tarafından yapıldığı anlaşılamamaktadır.

Genel prensip olarak, tanımlanan tüm delegelere hangi nesne tarafından çağrıldığını belirtir System.Object tipinde bir parametre eklenmesi önerilmektedir.

C#

. . .

**public** **delegate** **void** UsbDurumEventHandler(**object** sender, UsbDurum status);

**private** UsbDurumEventHandler durumDelegesi = **null**;

**public** **void** Aç() {

durum = UsbDurum.Açık;

// Çağrımda nesnenin kendisi de parametre olarak iletilmektedir.

// Böylece çağrımın hangi nesne tarafından yapıldığı anlaşılmaktadır.

**if** (durumDelegesi != **null**)

durumDelegesi(**this**, UsbDurum.Açık);

}

. . .

VB.NET

. . .

**Public** **Delegate** **Sub** UsbDurumEventHandler(**ByVal** sender **As** **Object**, \_  
 **ByVal** status **As** UsbDurum)

**Private** durumDelegesi **As** UsbDurumEventHandler = **Nothing**

**Public** **Sub** Aç()

durum = UsbDurum.Açık

' Çağrımda nesnenin kendisi de parametre olarak iletilmektedir.

' Böylece çağrımın hangi nesne tarafından yapıldığı anlaşılmaktadır.

**If** **Not** durumDelegesi **Is** **Nothing** **Then**

durumDelegesi(**Me**, UsbDurum.Açık)

**End** **If**

**End** **Sub**

. . .

UsbDurumEventHandler isimli delegenin değiştirilen parametre listesi ile, DurumGoster ve DurumLogla isimli metotlar içerisinde, çağrımın hangi nesne tarafından yapıldığı kolaylıkla anlaşılabilmektedir.

Önerilen delege parametre listesi şablonu, nesne referansı ile System.EventArgs sınıfı veya bu sınıfı miras alan bir sınıfı içermektedir.

C#

**public** **delegate** **void** TestDelegate(**object** sender, EventArgs e);

VB.NET

**Public** **Delegate** **Sub** TestDelegate(**ByVal** sender **As** **Object**, \_  
 **ByVal** e **As** EventArgs)

Kod 130, Önerilen Delege Şablonu

Yukarıdaki örnek şablonda;

* Çağrımın hangi nesne tarafından yapıldığı sender isimli parametre ile belirtilmekte.
* Bilgi iletimi ve değer geri alma işlemi e isimli parametre ile yönetilmektedir.

Bilgi iletimi veya değer geri alma ihtiyacı olmasa dahi yukarıdaki şablonun kullanımı önerilmektedir.

C#

**class** Test {

**public** **delegate** **void** TestDelegate (**object** sender, EventArgs e);

**public** TestDelegate delege;

**public** **void** TestMethod() {

// Herhangi bir bilgi iletimi veya değer alma ihtiyacı olmadığında

// EventArgs.Empty alanı kullanılmalıdır.

delege(**this**, EventArgs.Empty);

}

}

VB.NET

**Class** Test

**Public** **Delegate** **Sub** TestDelegate(**ByVal** sender **As** **Object**, \_  
 **ByVal** e **As** EventArgs)

**Public** delege **As** TestDelegate

**Public** **Sub** TestMethod()

' Herhangi bir bilgi iletimi veya değer alma ihtiyacı olmadığında

' EventArgs.Empty alanı kullanılmalıdır.

delege(**Me**, EventArgs.Empty)

**End** **Sub**

**End** **Class**

EventArgs.Empty alanı salt okunur bir alandır ve çağrımda herhangi bir bilgi iletim ihtiyacı olmadığında kullanılmalıdır.

EventArgs sınıfını miras alan yeni sınıflar tanımlayarak, önerilen delege parametre listesi şablonu ile bilgi iletimini sağlamak mümkündür.

C#

**class** USBPort {

. . .

// System.EventArgs sınıfını miras alan yeni bir sınıf tanımlanmıştır.

**public** **class** UsbDurumEventArgs: EventArgs {

**private** UsbDurum durum;

**public** UsbDurum Durum {

**get** {

**return** durum;

}

}

**public** UsbDurumEventArgs(UsbDurum durum) {

**this**.durum = durum;

}

}

**public** **delegate** **void** UsbDurumEventHandler(**object** sender,   
 UsbDurumEventArgs e);

**public** **void** Aç() {

durum = UsbDurum.Açık;

// Parametre olarak nesnenin kendisi ve UsbDurumEventArgs nesnesi   
 // iletilmektedir.

**if** (durumDelegesi != **null**)

durumDelegesi(**this**, **new** UsbDurumEventArgs(UsbDurum.Açık));

}

. . .

VB.NET

**Class** USBPort

. . .

' System.EventArgs sınıfını miras alan yeni bir sınıf tanımlanmıştır.

**Public** **Class** UsbDurumEventArgs

**Inherits** EventArgs

**Private** \_durum **As** UsbDurum

**Public** **Sub** **New**(**ByVal** durum **As** UsbDurum)

**Me**.\_durum = durum

**End** **Sub**

**Public** **ReadOnly** **Property** Durum() **As** UsbDurum

**Get**

**Return** \_durum

**End** **Get**

**End** **Property**

**End** **Class**

**Public** **Delegate** **Sub** UsbDurumEventHandler(**ByVal** sender **As** **Object**, \_  
 **ByVal** e **As** UsbDurumEventArgs)

**Public** **Sub** Aç()

durum = UsbDurum.Açık

' Parametre olarak nesnenin kendisi ve UsbDurumEventArgs nesnesi   
 ' iletilmektedir.

**If** **Not** durumDelegesi **Is** **Nothing** **Then**

durumDelegesi(**Me**, **New** UsbDurumEventArgs(UsbDurum.Açık))

**End** **If**

**End** **Sub**

. . .

Kod 131, System.EventArgs Sınıfı Kullanımı

Yukarıdaki örnekte UsbDurumEventArgs isminde EventArgs sınıfını miras alan yeni bir sınıf tanımlanmış ve delege çağrımında bu sınıf tipinde nesne oluşturulmuştur.

System.EventArgs sınıfını miras alan ve bilgi alış verişi için kullanılan sınıf isimlerinin EventArgs ile bitmesi bir zorunluluk olmamasına rağmen önerilen yöntemdir.

Aşağıda metot içerisinde örnek kullanım gösterilmiştir.

C#

**void** DurumGoster(**object** sender, USBPort.UsbDurumEventArgs e) {

Console.WriteLine("USB Port Durumu Değişti");

Console.WriteLine("Yeni Durum: {0}", e.Durum);

}

VB.NET

**Sub** DurumGoster( **ByVal** e **As** USBPort.UsbDurumEventArgs)

Console.WriteLine("USB Port Durumu Değişti")

Console.WriteLine("Yeni Durum: {0}", e.Durum)

**End** **Sub**

Herhangi bir bilgi iletimine ihtiyaç duyulmadığında System.EventHandler delegesi kullanılabilir. System.EventHandler tanımı aşağıda gösterilmiştir.  
  
C#:  
public delegate void EventHandler(object sender, EventArgs e);  
  
VB.NET:  
Public Delegate Sub EventHandler(ByVal sender As Object,   
 ByVal e As EventArgs)

Jenerik Delegeler

Delegeleri, parametre listesi jenerik olacak şekilde tanımlamak mümkündür.

C#

**public** **delegate** **void** TestDelegate<T>(**object** sender, T e) **where** T : EventArgs;

// Örnek Kullanım:

**public** TestDelegate<EventArgs> test;

VB.NET

**Public** **Delegate** **Sub** TestDelegate(**Of** T **As** EventArgs)(**ByVal** sender **As** **Object**,\_  
 **ByVal** e **As** T)

' Örnek Kullanım:

**Public** test **As** TestDelegate(**Of** EventArgs)

Listelenen örneklerde TestDelegate isimli delege jenerik olarak tanımlanmıştır. Jenerik sınıf tanımına benzer olan jenerik delege tanımı ile jenerik yapı kullanım avantajları elde edilmektedir.

System.EventHandler delegesinin jenerik sürümü aşağıda gösterilmiştir.

C#

**public** **delegate** **void** EventHandler<TEventArgs>(**object** sender, TEventArgs e) **where** TEventArgs: EventArgs

VB.NET

**Public** **Delegate** **Sub** EventHandler(**Of** TEventArgs **As** EventArgs)(**ByVal** sender **As** **Object**, **ByVal** e **As** TEventArgs)

Jenerik olan System.EventHandler delegesi, daha önce bahsedilen delege parametre listesi şablonu için rahatlıkla kullanılabilir. Böylece yeni delege tanımı yapmak yerine sadece System.EventArgs sınıfı veya bu sınıfı miras alan yeni bir sınıf tanımlayarak nesne haberleşmesi sağlanabilir.

C#

**class** UsbPort {

. . .

// Jenerik EventHandler kullanımı ile delege tanımı yapılmadan aynı işlevi   
 // elde etmek mümkündür.

**private** EventHandler<UsbDurumEventArgs> durumDelegesi = **null**;

. . .

}

VB.NET

**Class** UsbPort

. . .

' Jenerik EventHandler kullanımı ile delege tanımı yapılmadan aynı işlevi   
' elde etmek mümkündür.

**Private** EventHandler(**Of** UsbDurumEventArgs) **As** UsbDurumEventHandler = **Nothing**

. . .

**End** **Class**

Kod 132, Jenerik Delege Kullanımı

Anonim Metotlar

Anonim metotlar (ing. Anonymous Methods), delege tarafından çağrılması istenen metot yerine, belirli bir kod bloğunu belirtmek için kullanılmaktadır.

Anonim metot kullanımı aşağıda gösterilmiştir.

**class** Test {

**public** **delegate** **void** TestDelegate(**object** sender, EventArgs e);

**public** TestDelegate delege;

**public** **void** TestMethod() {

delege(**this**, EventArgs.Empty);

}

}

// Kullanım Örneği:

Test t = **new** Test();

// Anonim (İsimsiz) bir metot kullanılmaktadır.

t.delege += **delegate** {

Console.WriteLine("Test Delegate");

};

t.TestMethod();

Yukarıdaki örnekte delege parametreleri göz ardı edilmiştir. Delege parametrelerine erişim aşağıdaki örnekte olduğu gibi yapılabilir.

Test t = **new** Test();

t.delege += **delegate**(**object** sender, EventArgs e) {

// sender ve e isimli parametreler bu blok içerisinde kullanılabilir.

};

t.TestMethod();

Anonim metot kullanımı ile hedeflenen, daha okunabilir kod yapısı sağlamak ve az sayıda işlem için metot tanımlamadan aynı işlevi elde etmektir.

Anonim metotlar sadece C# tarafından desteklenmektedir. Blok sonlanması ardından eklenen ; karakterine dikkat ediniz.

Delege Sınıfı

Delegate rezerve kelimesi ile delege tanımlandığında, derleyici tarafından arka planda yapılan işlem System.MulticastDelegate sınıfını miras alan yeni bir sınıf tanımlamaktır.

Aşağıda System.MulticastDelegate üye bilgileri aşağıda gösterilmiştir.

| Üye Adı | Açıklama |
| --- | --- |
| Method | Delege tarafından saklanan metot ile ilgili bilgilere erişim amacıyla kullanılmalıdır. |
| Target | Delege tarafından saklanan metot statik ise null (VB.NET için nothing) değerini döndürür. Statik olmayan metotlar için metodun tanımlandığı nesneyi döndürür. |
| Combine | Delege tarafından tutulan metot listesine yeni bir metot eklemek için kullanılmalıdır. C# için += operatörü ile bu metot çağrılmaktadır. |
| Remove | Delege tarafından tutulan metot listesinden bir metodu çıkarmak için kullanılmalıdır. C# için -= operatörü ile bu metot çağrılmaktadır. |

Tablo 46, System.MulticastDelegate Sınıfı Üye Bilgileri

Olaylar

Delegeler, nesneler arası çift yönlü haberleşmeyi sağlayan yapılardır. Buna rağmen delege kullanımında, delegeye metot ekleme, çıkarma, gerekli delege değişkenlerini tanımlama gibi bir çok detay ile programcının ilgilenmesi gerekmektedir.

Olayları (ing. Events), delege kullanımını, dolayısı ile nesneler arası haberleşme işlevini, daha kolay şekilde yapabilmeyi sağlayan yapılar olarak tanımlamak mümkündür.

Event rezerve kelimesi ile yapılan olay tanımında derleyici tarafından yapılan işlem, delege kullanımında programcı tarafından tanımlanması gereken bazı metot ve değişkenleri otomatik olarak IL kod içerisinde oluşturmaktır.

C#

**class** USBPort {

. . .

// Delege tanımı.

**public** **delegate** **void** UsbDurumEventHandler(**object** sender,  
 UsbDurumEventArgs e);

// Olay tanımı aşağıda yapılmıştır.

**public** **event** UsbDurumEventHandler OnUsbDurumEvent;

**public** **void** Aç() {

durum = UsbDurum.Açık;

// Delege değişkeninde olduğu gibi, olay ile ifade edilen metotları çağır.

**if** (OnUsbDurumEvent != **null**)

OnUsbDurumEvent(**this**, **new** UsbDurumEventArgs(UsbDurum.Açık));

}

}

. . .

// Kullanım Örneği:

USBPort port = **new** USBPort();

// port.OnUsbDurumEvent olayına DurumGoster metodunu ekle.

port.OnUsbDurumEvent += DurumGoster;

port.Aç();

VB.NET

**Class** USBPort

. . .

' Delege tanımı.

**Public** **Delegate** **Sub** UsbDurumEventHandler(**ByVal** sender **As** **Object**, \_  
 **ByVal** e **As** UsbDurumEventArgs)

' Olay tanımı aşağıda yapılmıştır.

**Public** **Event** OnUsbDurumEvent **As** UsbDurumEventHandler

**Public** **Sub** Aç()

durum = UsbDurum.Açık

' RaiseEvent rezerve kelimesi, kendisine parametre olarak verilen olaya   
 ' bağlı metotların çağrımı amacıyla kullanılmaktadır.

**RaiseEvent** OnUsbDurumEvent(**Me**, **New** UsbDurumEventArgs(UsbDurum.Açık))

**End** **Sub**

. . .

**End** **Class**

' Kullanım Örneği:

**Dim** port **As** USBPort = **New** USBPort()

' AddHandler rezerve kelimesi port.OnUsbDurumEvent ile ifade edilen olaya   
' yeni bir metot eklemektedir. İşlemin tersi, RemoveHandler rezerve kelimesi

' ile yapılabilir.

**AddHandler** port.OnUsbDurumEvent, **AddressOf** DurumGoster

port.Aç()

Kod 133, Olay Tanımlama ve Kullanım Örneği

Olay kullanımının tercih edilmesiyle birlikte, delege kullanımında tanımlanması gereken yardımcı metotların (DurumEventHandlerEkle ve DurumEventHandlerÇıkar) tanımlanmasına gerek kalmadığına dikkat edilmelidir.

C# için olay kullanımı delege kullanımı ile oldukça benzerdir. += ve -= operatörleri delege kullanımında olduğu gibi olaya metot eklemek ve çıkarmak için kullanılmaktadır.

VB.NET için bir olaya metot eklemek ve çıkarmak sırasıyla AddHandler ve RemoveHandler rezerve kelimeleri ile yapılmaktadır. RaiseEvent rezerve kelimesi, parametre olarak iletilen olay tarafından tutulan metotların çağrımı amacıyla kullanılmaktadır.

VS.NET ile Olay Metotlarının Oluşturulması

VS.NET, bir olaya metot eklenmesini kolaylaştıran komutlar içermektedir.

Örneğin, olay adının ve += operatörünün kod editöründe yazılıp TAB tuşuna basılması ile otomatik olarak ilgili sınıf içerisinde metot oluşturulmaktadır.

Olayların Özellik Olarak Tanımlanması

NYP bilgi gizleme kuralları çerçevesinde, olay tanımlamalarını gizli alanlar - açık özellikler şeklinde yapmak mümkündür. Böylece, olaya yeni bir metot eklenmesi ve olaydan var olan bir metodun çıkarılması sırasında programcı tarafından istenilen işlemlerin yapılabilmesi mümkün olmaktadır.

C#

**class** Test {

// testEvent isminde gizli (private) olarak olay tanımı yapılmıştır.

**private** **event** EventHandler testEvent;

// Olay, özellik niteliğinde tanımlanmıştır.

**public** **event** EventHandler TestEvent {

**add** {

// Olaya yeni bir metot eklendiğinde çalıştırılacak blok.

Console.WriteLine("Ekleme");

testEvent += value;

}

**remove** {

// Olaydan var olan bir metot çıkarıldığında çalıştırılacak blok.

Console.WriteLine("Çıkarma");

testEvent -= value;

}

}

}

VB.NET

**Class** Test

' \_testEvent isminde gizli (private) olarak olay tanımı yapılmıştır.

**Private** **Event** \_testEvent **As** System.EventHandler

' Olay, özellik niteliğinde tanımlanmıştır.

**Public** **Custom** **Event** TestEvent **As** EventHandler

**AddHandler**(**ByVal** value **As** EventHandler)

' Olaya yeni bir metot eklendiğinde çalıştırılacak blok.

Console.WriteLine("Ekleme")

**End** **AddHandler**

**RemoveHandler**(**ByVal** value **As** EventHandler)

' Olaydan var olan bir metot çıkarıldığında çalıştırılacak blok.

Console.WriteLine("Çıkarma")

**End** **RemoveHandler**

**RaiseEvent**(**ByVal** sender **As** **Object**, **ByVal** e **As** System.EventArgs)

' Olay RaiseEvent ile çağrıldığında çalıştırılacak blok.

Console.WriteLine("Çağırma")

**End** **RaiseEvent**

**End** **Event**

**End** **Class**

Kod 134, Olayların Özellik Olarak Tanımlanması

Anonim Olaylar

VB.NET, olay tanımı yapılırken delege kullanımını zorunlu kılmamaktadır. Aşağıdaki örnekte olay tanımı yapılmasına rağmen, olayın ait olduğu delege tanım içerisinde yer almamaktadır.

**Class** Test

' Anonim olay tanımı aşağıda yapılmıştır.

**Public** **Event** TestEvent(**ByVal** s **As** **String**)

**Public** **Sub** TestMethod()

' Olaya bağlı metotları çağır.

**RaiseEvent** TestEvent("xyz")

**End** **Sub**

**End** **Class**

**Class** Program

**Shared** **Sub** EventMethod(**ByVal** s **As** **String**)

Console.WriteLine(s)

**End** **Sub**

**Shared** **Sub** Main(**ByVal** args() **As** **String**)

**Dim** t **As** Test = **New** Test()

' Anonim olaya yeni bir metot ekle.

**AddHandler** t.TestEvent, **AddressOf** EventMethod

t.TestMethod()

**End** **Sub**

**End** **Class**

VB.NET Otomatik Olay Yönetimi

VB.NET, WithEvents ve Handles rezerve kelimeleri ile daha kolay ve pratik şekilde olay yönetimi yapmayı sağlamaktadır.

**Class** Test

**Public** **Delegate** **Sub** TestEventHandler(**ByVal** i **As** **Integer**)

**Public** **Event** OnTestEvent **As** TestEventHandler

**Public** **Sub** TestMethod()

**RaiseEvent** OnTestEvent(12)

**End** **Sub**

**End** **Class**

**Class** Program

' testObj isminde nesne tanımı WithEvents rezerve kelimesi ile birlikte  
 ' yapılmıştır.

**Shared** **WithEvents** testObj **As** Test = **New** Test()

' testObj.OnTestEvent olayı Handles rezerve kelimesi ile otomatik olarak  
 ' aşağıdaki metoda bağlanmıştır.

**Shared** **Sub** EventMethod(**ByVal** i **As** **Integer**) **Handles** testObj.OnTestEvent

Console.WriteLine(i)

**End** **Sub**

**Shared** **Sub** Main(**ByVal** args() **As** **String**)

' Tek yapılması gereken ilgili metodu çağırmaktır. Olay yönetimi   
 ' VB.NET tarafından otomatik olarak yapılmaktadır.

testObj.TestMethod()

**End** **Sub**

**End** **Class**

Otomatik olay yönetimi olarak ifade edilen bu işlem sadece alanlar için yapılabilmektedir. Yerel değişkenler için AddHandler rezerve kelimesinin kullanımı zorunludur.  
  
Pratik kullanımda önemli avantajlar sağlamasına rağmen, metotların hangi sıra ile çağrılacağı belirli değildir. Metotların çağrım sırasının, ekleme sırası ile aynı olma zorunluluğunun bulunduğu durumlarda bu kullanım tercih edilmemelidir.

Gerçek mutluluk mal ve mülke sahip olmak ile değil, akıl ve erdeme sahip olmak ile mümkündür.

Aristoles

Bölüm

9

.NET Assembly Bilgileri

T

ekrar kullanılabilirlik kavramı ve sınıf kütüphaneleri, daha hızlı ve kaliteli yazılım elde etmek için kullanımı önerilen yöntemlerdendir.

Özellikle büyük çaplı yazılım projelerinde, bir uygulama birden fazla Assembly içermektedir. Assembly kullanımı, aynı zamanda, yazılımda mantıksal katmanların (Kullanıcı ara yüzü, iş katmanı, veri katmanı, konfigürasyon katmanı vb.) kullanılabilmesini ve farklı fiziksel yerleşimlere dağıtılabilmesini sağlamaktadır.

Bu bölümde Assembly kavramı, yapısı, konfigürasyonu ve kaynak kullanımı konularında bilgiler verilecektir.

Temel Bilgiler

Assembly, kendi içerisinde sürümü ve güvenlik yapısı ile bütün ikil dosyadır (ing. Binary File). .NET uygulamaları bir veya birden fazla Assembly dosyasından oluşabilmektedir.

Assembly dosya uzantılarının \*.dll veya \*.exe olabilmesi, bu dosyaların Win32 çalıştırılabilir dosyaları ile karıştırılmasına neden olmaktadır. Assembly dosyaları PE (Portable Executable) yapısında olmasına rağmen, bu benzerlik dışındaki tüm yapılar CLR tarafından yönetilmektedir.

Assembly adı, Assembly dosyasının uzantısı olmadan adıdır. Örneğin Assembly dosyası xyz.dll ise Assembly adı xyz'dir.

Ad Alanları ve Assembly

Kaynak kodun derleyici tarafından derlenmesi ile oluşturulan Assembly adı ile ad alanları (ing. Name Space) birbirlerine karıştırılabilen kavramlardır.

Şekil 21, Ad Alanları ve Assembly

Bir Assembly içerisinde istenilen sayıda ad alanı tanımlanabilir. Tanımlanan ad alanları ile Assembly adı arasında herhangi bir ilişki bulunmamaktadır.

Assembly içerisinde tanımlı ad alanları ile Assembly adı farklı olabilse de, genel prensip olarak üst ad alanı ile dosya adının aynı olması önerilmektedir. Örneğin, Assembly adı DotNetKitabı ise, bu Assembly içerisinde DotNetKitabı, DotNetKitabı.Matematik şeklinde tanımlama yapılması önerilmektedir.

Assembly Yapısı

Bir Assembly dosyası aşağıda gösterilen mantıksal bölümlerden oluşmaktadır.

Şekil 22, Assembly Dosya Yapısı

Assembly dosya yapısı ve bölümleri .NET Framework SDK içerisinde bulunan dumpbin.exe uygulaması ile analiz edilebilir.

Win32 Başlığı

Win32 başlığı, IL kod içermeyen standart Windows çalıştırılabilir dosyalarında da bulunan, dosyanın Windows tarafından çalıştırılabileceğini belirten başlıktır. Bu başlık aynı zamanda uygulamanın konsol veya Windows ara yüz uygulaması olduğunu da belirtmektedir.

Bu başlığın Assembly dosya içerisinde bulunmasının sebebi, dosyanın Windows platformunda çalıştırılmasıyla, CLR yüklemesinin yapılabilmesini sağlayan kodun saklanmasıdır.

CLR Başlığı

CLR tarafından analiz edilen ve Assembly hakkında çeşitli bilgilere erişimi sağlayan başlıktır.

IL Kodu

Kaynak kodun derleyici tarafından derlenmesi ile oluşturulan kod bu bölümde bulunmaktadır. IL kod, JIT tarafından çalışma zamanında yapılan ikinci bir derleme ile makine koduna çevrilmektedir.

Tip Bilgileri

IL kod içerisinde bulunan ve / veya Assembly tarafından referans verilen tipleri (Sınıflar, delegeler, arabirimler, vb.), bu tiplerin Assembly dosyası içerisindeki yerleşimlerini belirten bilgilerin bulunduğu bölümdür.

Assembly Bilgileri

Assembly sürüm bilgisi ve Assembly tarafından referans verilen diğer Assembly dosyaları gibi Assembly ile ilgili bilgilerin saklandığı bölümdür.

Gömülü Kaynaklar

Assembly dosyası, resim, video veya herhangi tipte farklı verileri içerisinde saklayabilmektedir. Kaynak olarak ifade edilen verilerin saklandığı bölümdür.

Assembly Kullanım Yöntemleri

Assembly kullanımı temel iki kategoride incelenebilir.

* Özel Assembly Kullanımı
* Paylaşımlı Assembly Kullanımı

Özel Assembly Kullanımı

Özel Assembly (ing. Private Assembly) kullanımı, uygulama tarafından referans verilen Assembly dosyalarının uygulamanın bulunduğu dizin içerisine kopyalanarak kullanımını belirtir kavramdır.

XCOPY dağıtım (ing. XCopy Deployment) olarak da anılabilen bu yöntem ile, uygulama tarafından referans verilen Assembly dosyalarının uygulama ile aynı veya uygulama klasörünün alt klasörüne kopyalanması dağıtım için yeterlidir. Uygulamanın sistemden kaldırılması sırasında kopyalanan Assembly dosyaları silinerek işlem tamamlanabilmektedir.

Şekil 23, XCopy Dağıtım

Derleme aşamasında, kaynak kod içerisinde using veya imports (sırasıyla C# ve VB.NET) ifadeleri ile bir ad alanı veya aynı VS.NET projesi içerisinde tanımlı olmayan bir tip kullanıldığında proje referansları içerisinde ad alanı veya tip araması yapılır. Benzer işlem çalışma zamanında da geçerlidir.

Özel Assembly Yükleme İşlemi

CLR, bir Assembly'yi yüklerken, Assembly içerisinde bir başka Assembly'ye referans bulduğunda, kaynak Assembly'nin bulunduğu klasör içerisinde, referans verilen dosyayı aramaktadır.

Aşağıda Assmbly dosyasının bilgiler bölümünde (Örneğin Uygulama.exe) bulunan örnek bir referans gösterilmiştir.

.assembly extern DotNetKitabı { ..}

Çalışma zamanında referans Assembly çözümü aşağıdaki sırayla yapılmaktadır.

1. İlk adım aynı klasör içerisinde DotNetKitabı.dll dosyasını aramaktır.
2. DotNetKitabi.dll isimli dosya bulunamaz ise DotNetKitabi.exe dosyası aranır.
3. Her iki dosya da bulunamazsa DotNetKitabı isimli alt klasör (eğer varsa) içerisinde aynı işlem yapılır.
4. Assembly hiçbir şekilde bulunamazsa FileNotFound istisnası fırlatılır.

Assembly kavramından sadece .dll uzantılı dosya anlaşılmamalıdır. .exe uzantılı dosyalar da (Assembly yapısına uyduğu sürece her türlü dosya uzantısı geçerlidir) aynı kurallar ile yönetilmektedir. Örneğin, Uygulama.exe içerisinden Kutuphane.exe isimli Assembly dosyasına referans vermek ve bu Assembly içerisindeki tiplere erişmek mümkündür.

Assembly Yükleme İşleminin Özelleştirilmesi

Uygulamanın dağıtımında özel Assembly kullanımı tercih edilmesine rağmen, uygulamadan referans verilen Assembly dosyalarının farklı klasörlere kopyalanması istenebilir. Örneğin;

C:\Program Files\Uygulama\Uygulama.exe  
C:\Program Files\Uygulama\Kütüphane\DotNetKitabı.dll  
C:\Program Files\Uygulama\Kütüphane\DotNetKitabı.Lib.dll

Bu durum Assembly konfigürasyon dosyası içerisinde tanımlanabilir.

Assembly konfigürasyon dosyası, Assembly ile ilgili tanımlamaların dışarıdan yapılabileceği metin bazlı dosyadır. Örneğin Assembly adı Uygulama.exe ise, konfigürasyon dosya adı Uygulama.exe.config olmalıdır. Konfigürasyon dosyaları ile ilgili bilgi ilerleyen konularda verilecektir.

<configuration>  
 <runtime>

<assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">

<**probing** privatePath="**Kütüphane**"/>

</assemblyBinding>

</runtime>

</configuration>

Yukarıdaki tanım ile, Uygulama.exe içerisinden referans verilen bir Assembly dosyasının aynı klasör içerisinde bulunan Kütüphane isimli alt klasörde aranması gerektiği belirtilmektedir.

; karakterleri ile ayrılmış klasör isimleri desteklenmektedir. Örneğin "Kütüphane;Kütüphane\Matematik" gibi.  
  
Belirtilen klasör sadece uygulama klasörü altında aranmaktadır.

Paylaşımlı Assembly Kullanımı

Paylaşımlı Assembly (ing. Shared Assembly) kullanımı ile sisteme bir Assembly dosyası yüklenerek ilgili Assembly dosyası birden fazla uygulama tarafından kullanılabilir (Paylaşılabilir).

.NET sınıf kütüphanesi bu kullanıma örnek olarak verilebilir. System.Web.Services.dll isimli Assembly dosyası.NET Framework ile birlikte sisteme yüklenmektedir. Bu Assembly dosyasına referans veren her uygulama aynı dosyayı kullanmaktadır.

Windows uygulaması geliştirme konusunda daha önce çalışmalar yapmış okuyucular, paylaşımlı Assembly kullanımını System32 klasörüne kopyalanan Win32 DLL dosyalarına benzetecektir. Önemli kurulum ve dağıtım problemlerini de beraberinde getiren paylaşımlı Win32 DLL kullanımı, .NET ile birlikte problemlerin ortadan kaldırıldığı bir mimariye kavuşmuştur.

Global Assembly Kaşesi – GAC

Paylaşımlı kullanılan Assembly dosyaları, global Assembly kaşesi (ing. Global Assembly Cache - GAC) içerisinde saklanmaktadır.

Şekil 24, Global Assembly Kaşesi

Ekran Çıktısı 26, GAC Penceresi

GAC içeriği, işletim sistemi Windows klasörü içerisinde bulunan Assembly klasörü içerisinde saklanmaktadır. Örneğin C:\Windows\Assembly klasörü.

GAC içerisinde, isimleri ve uzantıları aynı olan ancak sürümleri birbirinden farklı olan Assembly dosyaları saklanabilmektedir. Yukarıdaki ekran çıktısında, üç adet Advantage.Data.Provider isimli Assembly görülmektedir. Aynı isimde ve aynı açık anahtar (ing. Public Key) değerine sahip olmalarına rağmen sürümlerinin birbirlerinden farklı olduğuna dikkat edilmelidir.

GAC içerisine sadece güçlü isimlendirilmiş Assembly dosyaları yüklenebilir. Güçlü isimlendirilmiş Assembly kavramı hakkında ilerleyen konularda detaylı açıklama yapılacaktır.

Global Assembly Kaşesi Yönetimi

GAC yönetimi görsel olarak .NET Konfigürasyon Aracı (.NET Configuration Utility) ile veya bir konsol uygulaması olan gacutil.exe yardımıyla komut satırından (ing. Command Prompt) yapılabilir.

.NET Konfigürasyon Aracı

.NET konfigürasyon aracı, sistem genelinde .NET Framework ayarlarının yapılacağı görsel kontrol panel uygulamasıdır (ing. Control Panel Applet).

Kontrol Panel | Yönetimsel Araçlar (Control Panel | Administrative Tools) içerisinde bulunan konfigürasyon aracı aşağıda gösterilmiştir.

Ekran Çıktısı 27, .NET Konfigürasyon Aracı

Var olan bir Assembly dosyasını GAC'a eklemek için Add komutu verilmelidir. Yüklü olan bir Assembly üzerinde sağ tıklanıp Delete komutu ile seçili Assembly GAC'tan kaldırılabilir. Seçili Assembly üzerinde sağ tıklayıp Properties komutu yardımı ile Assembly özellikleri görüntülenebilir.

Ekran Çıktısı 28, Assembly Özellikleri Penceresi

Komut Satırı Yardımıyla GAC Yönetimi

.NET Framework ile dağıtılan gacutil.exe aracı yardımıyla komut satırından GAC yönetimi yapmak mümkündür. Uygulama /? komut satır parametresi ile çalıştırılarak desteklenen parametreler hakkında daha detaylı bilgi alınabilir.

Aşağıda bazı komut satırı parametreleri ve açıklamaları gösterilmiştir.

| Parametre | Açıklama |
| --- | --- |
| / l | GAC içerisindeki Assembly listesini görüntülemek için kullanılmalıdır. |
| / i | GAC içerisine var olan Assembly dosyasını yüklemek için kullanılmalıdır. |
| / u | GAC içerisinde yüklü olan Assembly dosyası kaldırmak için kullanılmalıdır. |

Tablo 47, GacUtil.exe Komut Satırı Parametreleri

Bir Assembly hakkında bilgi almak için /l parametresi Assembly adı ile birlikte kullanılabilir. Örneğin

> gacutil.exe /l Advantage.Data.Provider

The Global Assembly Cache contains the following assemblies:

Advantage.Data.Provider, Version=8.10.2.0, Culture=neutral, PublicKeyToken=e33137c86a38dc06, proce

ssorArchitecture=MSIL

Advantage.Data.Provider, Version=8.0.0.0, Culture=neutral, PublicKeyToken=e33137c86a38dc06

Advantage.Data.Provider, Version=8.10.1.0, Culture=neutral, PublicKeyToken=e33137c86a38dc06

Number of items = 3

Güçlü İsimlendirme

Güçlü isimlendirme (ing. Strong Naming), farklı özellik ve nitelikler yardımıyla bir Assembly dosyasının tekil hale getirilmesine verilen isimdir. Güçlü isimlendirilmiş Assembly (ing. Strong Named Assembly), güçlü isimlendirme yöntemlerinin uygulandığı Assembly'dir.

Güçlü isimlendirme ile ad alanları birbirine karıştırılmamalıdır.  
  
  
  
Örneğin, aynı sınıf ismi farklı Assembly dosyalarında tanımlı olabilir. Bu durumda hangi sınıfın kullanılmak istendiği ad alanları yardımıyla bütünleşik isim nitelendirmesi ile belirtilebilir.

Güçlü isimlendirilmiş Assembly kullanımı ile hedeflenen temel amaçlar aşağıda listelenmiştir.

* Doğru sürüm numarasına sahip Assembly'nin kullanılması. System32 klasöründe saklanan Standart Win32 DLL dosyalarında yaşanan en büyük sıkıntılardan birisi de sürüm yönetimidir. Güçlü isimlendirilmiş Assembly kullanımı ile bu problem ortadan kalmıştır.  
  Örneğin A isimli Assembly, B isimli Assembly dosyasına referans veriyor ise, referans bilgisi, sürüm bilgisini de içerebilmektedir. Böylece, B isimli Assembly dosyasının yeni sürümü sisteme yüklenmiş olsa dahi, referans eski sürümü belirttiği için olası sürüm uyumsuzluğu problemi çözülmüş olmaktadır.
* Güvenlik. Referans veren Assembly içerisinde, referans verilen Assembly ile ilgili güvenlik bilgileri saklanmaktadır. Böylece kötü niyetli bir kullanıcının Assembly üzerinde yapacağı değişiklikler geçersiz olmaktadır.  
  Örneğin, kötü niyetli bir kullanıcı System.Security.dll isimli Assembly dosyasının orijinal sürümünü sistemden kaldırıp, kendi geliştirdiği aynı isimli Assembly dosyasını GAC'a yükleyebilir. Bu durumda System.Security.dll isimli Assembly'yi kullanan tüm uygulamalar başarıyla çalışabilse de, hassas bilgiler kötü niyetli kişi tarafından ele geçirilebilmektedir.

Güçlü isimlendirme aşağıda listelenen verilerin birleşmesi ile oluşmaktadır.

1. Assembly adı.
2. Assemby sürüm bilgisi
3. Açık anahtar (ing. Public Key).
4. Dijital imza
5. Kültür bilgisi (İsteğe bağlı)

Listelenen verilerin birleşmesi ile bir Assembly tekil olarak ifade edilebilmektedir.

Aşağıda güçlü isimlendirilmiş Assembly referans örneği gösterilmiştir.

Advantage.Data.Provider, Version=8.10.2.0, Culture=neutral, PublicKeyToken=E33137C86A38DC06

Tanım ile, Advantage.Data.Provider isimli, 8.10.2.0 sürümüne sahip, kültür bilgisi olmayan ve açık anahtar özeti E33137C86A38DC06 olan Assembly dosyası referans verilmektedir.

**PKI ve Dijital İmza Hakkında Özet Bilgi**  
  
Açık anahtar altyapısı (ing. Public Key Infrastructure – PKI), matematiksel olarak birbirleri ile ilişkili iki bilgiyi esas almaktadır. Bilgiden kasıt belirli uzunluktaki ikil değerlerdir.  
  
Gizli anahtar (ing. Private Key) ve açık anahtar (ing. Public Key) olarak tanımlanan bu iki değer birbirleri ile matematiksel olarak ilişkilidir. Asimetrik şifreleme algoritmaları (ör. RSA) yardımıyla, gizli anahtar ile şifrelenen veri sadece açık anahtar ile, açık anahtar ile şifrelenen veri sadece gizli anahtar ile çözülebilmektedir.   
  
Gizli anahtar sadece üretici kuruluşa özgüdür ve başkalarıyla paylaşılmamaktadır. Açık anahtarın paylaşımında herhangi bir sakınca bulunmamaktadır.  
  
Özet bilgi (ing. Hash), özet bilgi algoritmaları (MD5, SHA, vb.) yardımıyla veriyi işleyerek sadece ilgili veri ile eşleşebilen daha küçük boyutlu veriyi ifade etmektedir. Orijinal veride olabilecek herhangi bir değişiklikte özet bilgi de değişmektedir.  
  
Dijital imza ile hedeflenen herhangi bir verinin kime ait olduğunun ve orijinal halinin korunduğunun inkar edilemez şekilde belirtilmesidir. Bu amaçla, veriye ait özet bilgi hesaplanmakta, özet bilgi gizli anahtar ile şifrelenmekte ve oluşan imza (Özet bilginin gizli anahtar ile şifrelenmiş hali) veri ile birlikte dağıtılmaktadır. Veriyi alan kişi, aynı özet algoritması ile özet bilgiyi hesaplamaktadır. Daha sonra, veri ile dağıtılan imza bilgisini açık anahtar ile deşifre etmektedir. Her iki bilgi (Özet bilgi ve deşifre edilen bilgi) birbirleri ile eşit ise veri doğru kişi tarafından oluşturulmuş ve orijinal hali korunmuştur.  
  
Kitabın ilerleyen bölümlerinde PKI kavramı daha detaylı şekliyle aktarılacaktır.

Aşağıda güçlü isimlendirilmiş Assembly oluşturma algoritması gösterilmiştir.

Şekil 25, Güçlü İsimlendirme Algoritması

1. Kaynak kod derleyici tarafından derlenmekte ve Assembly yapısı oluşturulmaktadır. Derleme aşamasında açık anahtar bilgisi Assembly içerisine gömülmektedir.
2. Assemby dosyasının özet bilgisi hesaplanmaktadır.
3. Hesaplanan özet bilgi programcı tarafından belirtilen gizli anahtar ile şifrelenmekte, böylece dijital imza oluşturulmaktadır.
4. Oluşturulan dijital imza bilgisi Assembly içerisine gömülmektedir.

Güçlü İsimlendirme Örneği ve Temel İşlemler

Bu bölümde, bir VS.NET sınıf kütüphanesi projesine güçlü isimlendirme işleminin nasıl uygulanacağı konusunda bilgiler verilecektir. Bu amaçla aşağıdaki sınıf gerçekleştirimi yapılmıştır.

| C# | VB.NET |
| --- | --- |
| **using** System;  **using** System.Collections.Generic;  **using** System.Text;  **using** System.Diagnostics;  **namespace** DotNetKitabı.Lib {  **public** **class** WebYöneticisi {  **static** **void** UrlAç(**string** url) {  ProcessStartInfo info;  info = **new** ProcessStartInfo();  info.UseShellExecute = **true**;  info.FileName = url;  Process.Start(info);  }  }  } | **Imports** System  **Imports** System.Collections.Generic  **Imports** System.Text  **Imports** System.Diagnostics  **Namespace** DotNetKitab.Lib  **Public** **Class** WebYöneticisi  **Shared** **Sub** UrlAç(**ByVal** url **As** **String**)  **Dim** info **As** ProcessStartInfo  Info = **New** ProcessStartInfo  Info.UseShellExecute = **True**  Info.FileName = url  Process.Start(info)  **End** **Sub**  **End** **Class**  **End** **Namespace** |

Statik WebYöneticisi.UrlAç metodu, url parametresi ile belirtilen web adresini sistemde tanımlı internet tarayıcısı ile açmaktadır.

**1. Assembly Adının Belirlenmesi**

Ön tanımlı Assembly adı VS.NET proje adı, yani proje dosyası (\*.csproj ve \*.vbproj) ismidir. VS.NET Proje Özellikleri | Uygulama (Project Properties | Application) sayfası yardımıyla çıktı Assembly adı düzenlenebilir. Bu sayfa yardımıyla, belirtilmediği durumlarda kullanılması istenen ad alanı tanımını da düzenlemek mümkündür.

Ekran Çıktısı 29, Assembly Adı ve Ön Tanımlı Ad Alanı

**2. Assembly Sürüm ve Kültür Bilgisinin Düzenlenmesi**

Assembly sürüm ve kültür bilgisi sırasıyla AssemblyVersionAttribute ve AssemblyCulture öznitelikleri ile veya VS.NET Proje Özellikleri | Uygulama | Assembly Özellikleri sayfası yardımıyla düzenlenebilir.

Ekran Çıktısı 30, Assembly Sürüm ve Kültür Bilgisi

Assembly öznitelikleri ile ilgili detaylı bilgi kitabın ilerleyen bölümlerinde verilecektir.

**3. Gizli ve Açık Anahtar İkilisinin Oluşturulması**

Gizli – açık anahtar ikilisinin oluşturulması VS.NET Proje Özellikleri | İmzalama (Project Properties | Signing) sayfası ile görsel olarak veya .NET Framework SDK ile dağıtılan sn.exe konsol uygulaması ile komut satırından yapılabilir.

Ekran Çıktısı 31, Açık - Gizli Anahtar İkilisinin Oluşturulması

Aynı işlem sn -k publicprivate.snk komutu ile de yapılabilmektedir.

İşlemlerin başarıyla tamamlanması sonucunda, içerisinde gizli ve açık anahtar bilgilerinin bulunduğu dosya oluşturulacaktır.

**4. Açık – Gizli Anahtar Dosyasının Belirtilmesi**

Anahtar dosya yerleşimi AssemblyKeyFileAttribute öz niteliği ile veya proje özellikleri sayfası ile belirtilebilir. Dosya yerleşiminin sadece proje özellikleri sayfası ile belirtilmesi önerilmektedir.

Ekran Çıktısı 32, Gizli - Açık Anahtar Dosyasının Belirtilmesi

**5. Derleme**

Adımların tamamlanması ardından projenin derlenmesi ile DotNetKitabı.Lib.dll isimli güçlü isimlendirilmiş Assembly derleyici tarafından oluşturulacaktır.

**6. Global Assembly Kaşesine Yükleme**

Gacutil.exe –i komutu ile Assembly dosyası GAC içerisine yüklenebilir.

Ekran Çıktısı 33, Oluşturulan Assembly Dosyası ve GAC

**7. Referans Verme**

VS.NET çözüm penceresi içerisinde proje üzerine sağ tıklanarak Add Reference ... komutu yardımıyla GAC içerisinde bulunan DotNetKitabı.Lib isimli Assembly'ye projeden referans verilebilir.

**8. Açık Anahtar Özet Bilgisinin Alınması**

Assembly referans bilgisi aşağıdaki gibidir.

Advantage.Data.Provider, Version=8.10.2.0, Culture=neutral, PublicKeyToken=E33137C86A38DC06

Bilgi, virgül ile birbirinden ayrılmış Assembly adı, sürümü, kültürü ve Assembly içerisinde bulunan açık anahtarın özet bilgisinden oluşmaktadır. Açık anahtarın kendisi yerine PublicKeyToken olarak tanımlanan 8 Byte uzunluğundaki özet bilgisinin kullanım amacı, referans uzunluğunun çok fazla olmaması ve daha kolay verilebilmesini sağlamaktır.

Açık anahtar özet bilgisi aşağıdaki şekilde elde edilebilir.

Şekil 26, Açık Anahtar Özet Bilgisinin Elde Edilmesi

Geciktirilmiş İmzalama

Assembly dosyasının imzalanması için gizliliği korunması gereken gizli – açık anahtar dosyası gerekmektedir. Üretici bazında olan ve Assembly'nin ilgili üretici tarafından dağıtıldığının inkar edilemez şekilde kanıtı olan bu dosyanın tüm ekip üyeleri ile paylaşımı istenmeyebilir.

Geciktirilmiş imzalama (ing. Delayed Signing), gizli anahtar yerine açık anahtar ile kısmi imzalamayı (ing. Partial Signing) ifade etmektedir.

Geciktirilmiş imzalama yapmak için proje özellikleri sayfasında Delay Sign Only işaretlenmeli veya aşağıdaki adımlar izlenmelidir.

1. Açık anahtar sn.exe –p komutu ile elde edilmelidir.
2. AssemblyKeyFile öz niteliği ile açık anahtar dosyası belirtilmeli ve AssemblyDelaySign öz niteliği True olarak belirtilmelidir.
3. sn.exe –Vr DotNetKitabı.Lib.dll komutu ile doğrulama etkinsizleştirilmelidir.
4. Geliştirme ve test etme süreci tamamlandıktan sonra   
   sn.exe – R DotNetKitabı.Lib.dll PublicPrivate.snk komutu ile Assembly imzalanmalıdır.

Assembly Yönetimi

Bu konu başlığında, aktarılan bilgiler ışığında Assembly konfigürasyonu ve yönetimi hakkında bilgiler verilecektir.

Assembly Referansları Hakkında

Bilindiği gibi referans bilgileri Assembly dosya yapısı içerisinde Assembly Bilgileri (Assembly Manifest) bölümünde saklanmaktadır. Referans verilen Assembly dosya yerleşimleri derleme aşamasında derleyiciye /reference komut satırı parametresi yardımıyla iletilmektedir.

csc KaynakDosya.cs /reference: DotNetKitabı.Lib.dll  
vbc KaynakDosya.vb /reference: DotNetKitabı.Lib.dll

VS.NET çözümü (ing. Solution) içerisinde birden fazla proje olması durumunda, Add Reference ... komutu ile proje bazında referans verilmesi mümkündür. Bu durumda, çözümün derlenmesi aşamasında VS.NET tarafından yapılan işlem, projelerin birbirlerine olan bağımlılıklarının analiz edilmesidir. Gerekli analiz yapıldıktan sonra uygun sıra ile projeler derlenmektedir.

Ekran Çıktısı 34, VS.NET Proje Referans Bilgisi

Proje referansları içerisinde seçili referans bilgisine ait Copy Local özelliğinin True olması, referans projenin başarıyla derlenmesi sonucu üretilen çıktının proje klasörüne kopyalanacağını belirtmektedir. Değerin False olarak belirlenmesi, referansın bulunduğu proje klasörünün kullanılması gerektiğini ifade etmektedir.

Uygulama Konfigürasyon Bilgileri

Uygulamaların konfigürasyon dosyaları yardımıyla yapılandırılması mümkünüdür. Özel Assembly kullanımı konusunda bu işlemle ilgili bir örnek gösterilmişti.

Uygulama Konfigürasyon Dosyaları

Uygulama konfigürasyon dosyaları uzantıları .config olan XML yapısında olan dosyalardır. VS.NET ile Add New Item ... komutu ile projeye konfigürasyon dosyası eklenebilir.

Ekran Çıktısı 35, Projeye Uygulama Konfigürasyon Dosyası Eklenmesi

Derlemenin yapılması ardından App.Config isimli dosya proje çıktı klasörüne (Örneğin \Debug\Bin veya \Release\Bin) ConsoleApplication5.exe.config olarak kopyalanacaktır.

Konfigürasyon dosyaları yardımıyla uygulama ile ilgili derleme ve çalışma zamanında bazı ayarlamaların yapılması mümkündür. Kitabın ilerleyen bölümlerinde uygulama konfigürasyon yönetimi hakkında detaylı açıklamalar yapılacaktır.

Uygulama konfigürasyon dosyaları yönetiminin yapılabileceği sınıflar System.Configuration.dll isimli Assembly içerisinde bulunmaktadır.

.NET Konfigürasyon Aracı

Uygulamaların veya Assembly'lerin .NET Konfigürasyon Aracı ile yapılandırılması mümkündür.

Ekran Çıktısı 36, Uygulamanın .NET Konfigürasyon Aracı ile Yapılandırılması

Örneğin, özel Assembly kullanımı bölümünde aktarılan probing ayarının konfigürasyon aracı ile yapılması mümkündür. Uygulama üzerinde sağ tıklanıp Properties komutu ile açılan pencerede gerekli ayarların girilmesi yeterlidir.

Ekran Çıktısı 37, Konfigürasyon Aracı ile Probing Ayarı

İşlemin onaylanması ile konfigürasyon dosyası içerisine gerekli bilgiler otomatik olarak araç tarafından eklenecektir.

<assemblyBinding   
 xmlns="urn:schemas-microsoft-com:asm.v1">

<**probing**   
 privatePath="**Lib;Kütüphane;Kütüphane\Diğer**"

/>

</assemblyBinding>

Bir Assembly dosyasının uygulama bazında konfigürasyonu Configured Assemblies bölümüne ilgili Assembly dosyasını ekleyerek yapılabilir. Konfigürasyon aracı ile ilerleyen konularda teorik düzeyde aktarılacak işlemlerin de görsel olarak yapılması mümkündür.

Konfigürasyon aracı tanımlamaların görsel olarak yapılabilmesini sağlamaktadır. Aynı işlem konfigürasyon dosyalarının elle düzenlenmesi ile de yapılabilir.

Sürüm İlkesi Yönetimi

Uygulama tarafından bir Assembly dosyasına referans verilmesi durumunda, ilgili referans bilgisi içerisinde referans verilen Assembly sürümü de saklanmaktadır. Sürüm ilkesi (ing. Binding Policy) yardımıyla farklı sürüm Assembly'lere yönlendirme yapmak mümkündür.

Örnek uygulama konfigürasyon dosyası içeriği aşağıda gösterilmiştir.

<configuration>

<runtime>

<assemblyBinding   
 xmlns="urn:schemas-microsoft-com:asm.v1">

<dependentAssembly>

<assemblyIdentity   
 name="DotNetKitabı.Lib"   
 publicKeyToken="c47eb22ef52313bf" />

<**bindingRedirect**   
 **oldVersion**="**1.2.0.0**"   
 **newVersion**="**2.0.0.0**"   
 />

</dependentAssembly>

</assemblyBinding>

</runtime>

</configuration>

Yukarıda yapılan işlem, uygulamanın 1.2.00 sürümüne sahip Assembly dosyasını referans göstermesine rağmen, çalışma zamanında otomatik olarak 2.0.0.0 sürümünün kullanımını sağlamaktır.

Eski sürüm bilgisi (oldVersion) aralık olarak belirtilebilir. Örneğin 1.0.0.0 – 1.9.9.9 gibi. Bu durumda 1.\* sürümüne referans veren uygulama otomatik olarak 2.0.0.0 sürümünü kullanacaktır.  
  
Aynı işlem .NET Konfigürasyon Aracı yardımıyla aşağıdaki şekilde yapılabilir.  
  
1. Yapılandırılmak istenen uygulama belirlenir.  
2. Configured Assemblies bölümüne ilgili Assembly eklenir.   
3. Binding Policy sekmesi aracılığı ile eski sürüm – yeni sürüm yönlendirme ayarları yapılır.

Assembly Fiziksel Yerleşimi

Uygulama tarafından referans verilen Assembly'ye ait fiziksel yerleşimin düzenlenmesi mümkündür.

Örnek uygulama konfigürasyon dosyası içeriği aşağıda gösterilmiştir.

<configuration>

<runtime>

<assemblyBinding . . .>

<dependentAssembly>

<assemblyIdentity   
 name="DotNetKitabı.Lib"   
 publicKeyToken="c47eb22ef52313bf" />

<**codeBase**   
 version="**2.0.0.0**"   
 href="**file:///C:\Lib\Kitap.dll**" />

</dependentAssembly>

</assemblyBinding>

</runtime>

</configuration>

Böylece uygulama tarafından referans verilen 2.0.0.0 sürümü farklı bir klasör içerisinde farklı bir isimle belirtilebilmektedir.

Codebase olarak tanımlanan bu işlem uzak fiziksel yerleşimleri de desteklemektedir.

<dependentAssembly>

<assemblyIdentity   
 name="DotNetKitabı.Lib"   
 publicKeyToken="c47eb22ef52313bf" />

<codeBase   
 version="2.0.0.0"   
 href="file:///C:\Lib\Kitap.dll" />

<**codeBase**   
 version="**3.0.0.0**"   
 href="**http://www.dotnetturk.com/DotNetKitabı.dll**"/>

</dependentAssembly>

Yukarıda yapılan tanımlama ile 3.0.0.0 sürümüne sahip Assembly belirtilen web adresinden otomatik olarak indirilecek ve kullanıma hazır olacaktır.

Uzak sistemlerden indirilen Assembly dosyaları indirme kaşesi (ing. Download Cache) olarak ifade edilen yerleşimde saklanmaktadır. İndirme kaşesi içeriği gacutil.exe /ldl komutu ile görüntülenebilir.

İşlemin .NET Konfigürasyon Aracı ile Codebases sekmesi yardımıyla görsel olarak yapılması mümkündür.

Assembly Konfigürasyon Bilgileri

Farklı uygulamalar tarafından kullanılabilen ve GAC içerisine yükleniş olan paylaşımlı Assembly dosyalarının sistem genelinde konfigürasyonunun yapılması mümkündür.

Bir önceki konuda uygulama bazında aktarılan işlemlerin geçerli olabilmesi için her uygulama klasöründe ilgili uygulama konfigürasyon dosyasının bulunması gereklidir. Sistemde yüklü uygulamaların sayısının artması ile işlemlerin yönetimi zorlaşabilmektedir.

GAC genelinde Assembly yönetimi, üretici bazında veya tüm sistem bazında ilkeler tanımlayarak yapılabilir.

Üretici Bazlı İlke Yönetimi

Üretici bazlı ilke yönetimi ile, üretici tarafından dağıtılan Assembly dosyalarının tek elden yapılandırılması mümkündür. Üretici İlkesi (ing. Publisher Policy) olarak tanımlanan bu işlem ile, üreticinin tüm ürünleri için tek konfigürasyon yeterli olmaktadır. Aşağıda örnek üretici ilke dosyası gösterilmiştir.

<configuration>

<runtime>

<assemblyBinding  
 xmlns="urn:schemas-microsoft-com:asm.v1">  
 <dependentAssembly>

<assemblyIdentity   
 name="DotNetKitabı.Lib"   
 publicKeyToken="c47eb22ef52313bf" />

<bindingRedirect oldVersion="1.2.3.4"

newVersion="2.0.0.0"/>

<codeBase version="3.0.0.0" href="**. . .**"/>

</dependentAssembly>

</assemblyBinding>

</runtime>

</configuration>

Üretici ilke Assembly (Publisher Policy Assembly), yukarıda gösterilen üretici ilkesi dosyasının derlenmesi ile oluşturulan Assembly'ye verilen isimdir. .NET Framework SDK ile dağıtılan al.exe yardımıyla üretici ilke Assembly dosyaları oluşturmak mümkündür.

Üretici ilke Assembly dosyasının oluşturulabilmesi için üreticiye ait gizli – açık anahtar dosyası (.snk) gerekmektedir. Böylece, üretici tarafından yapılandırılan Assembly'ler ile üretici ilke Assembly'sinin aynı üretici tarafından dağıtıldığı kesinlik kazanmaktadır.

Al.exe /link:[Üretici İlke Dosyası]  
 /out :[Üretici İlke Assembly Dosyası]  
 /keyfile: [Üreticiye Ait Anahtar Dosyası]  
 /platform: [İşlemci Mimarisi]

Üretici ilke Assembly dosya adı üst sürüm ve alt sürüm bilgilerini de içerecek şekilde aşağıdaki örnekte olduğu gibi belirtilmelidir.

Aşağıda örnek kullanım gösterilmiştir.

Al.exe /link:policy.config  
 /out :policy.1.0.DotnetKitabı.Lib.dll  
 /keyfile: PublicPrivate.snk  
 /platform: MSIL

Komut yardımıyla oluşturulan policy.1.0.DotnetKitabı.Lib.dll isimli gacutil.exe /i komutu ile GAC'a yüklenebilir.

Çalışma zamanında herhangi bir uygulama tarafından DotNetKitabı.Lib.dll Assembly dosyasının yüklenmesi aşamasında, CLR tarafından bu Assembly'ye ait üretici ilkeleri okunmakta ve gerekli yönlendirme yapılmaktadır. Üreticinin yayınlayacağı yeni ilke tanımları sürümleri artırılarak GAC'a yüklenmelidir.

Üretici ilkelerini, uygulama veya sistem bazında etkinsizleştirmek mümkündür.

<configuration>

<runtime>

<assemblyBinding   
 xmlns="urn:schemas-microsoft-com:asm.v1">

<**publisherPolicy**   
 apply="**no**" />

</assemblyBinding>

</runtime>

</configuration>

Sistem Bazlı İlke Yönetimi

Sistem bazlı ilke yönetimi (ing. Machine Policy veya Administrator Policy) ile, Assembly konfigürasyon yönetimin sistem bazında yapılması mümkündür. Bu şekilde yapılan konfigürasyon tanımlamaları ön tanımlı olarak ifade edilebilir. Bir başka anlatımla, Assembly yüklenmesi sırasında öncelikle uygulama bazlı tanımlar, daha sonra üretici bazlı tanımlar ve son olarak sistem bazlı tanımlar geçerlidir.

Sistem yöneticisi, bazı konfigürasyon ayarlarının uygulama veya üretici bazında yapılmasını engelleyebilmektedir. Böyle bir durumda çalışma zamanında işlemin geçersiz olduğunu belirtir istisna fırlatılacaktır.

Sistem bazlı ilke yönetimi, fiziksel olarak Windows\Microsoft.NET\[.NET Framework Sürümü]\Config içerisinde bulunan machine.config isimli dosya ile yapılmaktadır. İçerisinde bir çok konfigürasyon tanımının bulunduğu bu dosya üzerinde yapılacak hatalı düzenlemeler CLR'nin çalışmaması ile sonuçlanacaktır. Dosya üzerinde düzenleme yapılmadan önce yedeğinin alınması önerilmektedir.

Aşağıda machine.config dosyasının bir bölümü gösterilmiştir.

<runtime>

<assemblyBinding   
 xmlns="urn:schemas-microsoft-com:asm.v1">

<dependentAssembly>

<assemblyIdentity   
 name="DotNetKitabı.Lib"   
 publicKeyToken="c47eb22ef52313bf" />

<bindingRedirect   
 oldVersion="1.0.0.0 - 1.9.9.9"   
 newVersion="2.0.0.0" />

</dependentAssembly>

</assemblyBinding>

</runtime>

Yapılan işlem, sistem genelinde, DotNetKitabı.Lib isimli Assembly'nin yüklenmesinde 1.\* sürümü yerine 2.0.0.0 sürümünün yüklenmesini sağlamaktır.

.NET Konfigürasyon Aracı ile, herhangi bir uygulama üzerinde işlem yapılmadan, GAC içerisinde bulunan Assembly direkt Configured Assemblies bölümüne eklenerek gerekli ayarlamalar yapılabilir.

Araç ile yapılacak düzenlemeler machine.config dosyasına yazılmaktadır.

Ekran Çıktısı 38, Sistem Bazında Assembly Konfigürasyonu

Assemby Öznitelikleri ve Sürüm Yönetimi

Proje özellikleri sayfası ile telif hakkı, sürüm bilgileri gibi çıktı Assembly özelliklerinin yönetimi konusunda kitabın daha önceki konularında bilgiler verilmişti. Bu özellikler AssemblyInfo.cs ve AssemblyInfo.vb (sırasıyla C# ve VB.NET) isimli dosyada öznitelikler yardımıyla saklanmaktadır.

Ekran Çıktısı 39, Assembly Bilgileri Kaynak Dosyası

Proje özellikleri sayfası ile yapılan düzenlemeler otomatik olarak bu dosyaya yansıtılmaktadır.

Öznitelik Açıklamaları

Aşağıda temel Assembly öznitelikler ve açıklamaları listelenmiştir.

| Öznitelik | Açıklama |
| --- | --- |
| AssemblyTitle | Assembly başlık bilgisini belirtir özniteliktir. |
| AssemblyDescription | Assembly ile ilgili açıklamayı belirtir özniteliktir. |
| AssemblyConfiguration | Assembly'ye karakter dizi tipinde programcı tarafından herhangi bir konfigürasyon değeri vermek amacıyla kullanılmalıdır. Bu değer CLR tarafından göz ardı edilmektedir. |
| AssemblyCulture | Assembly kültür bilgisini belirtir özniteliktir. Herhangi bir değer verilmemesi durumunda Assembly kültür bilgisi içermez. |
| AssemblyCompany | Assembly üretici firma adını belirtir özniteliktir. |
| AssemblyProduct | Ürün (Program veya kütüphane) adını belirtir özniteliktir. |
| AssemblyCopyright | Assembly ile ilgili telif hakkı bilgisini belirtir özniteliktir. |
| ComVisible | Assembly ve içerisinde tanımlı tiplerin COM (Component Object Model) görünebilirliğini belirtir özniteliktir. |
| Guid | Assembly'nin COM görünürlüğünün bulunması durumunda kullanılan evrensel tekil belirteçtir (ing. Global Unique Identifier). |

Tablo 48, Assembly Öznitelik Bilgileri

Assembly özelliklerinin AssemblyInfo isimli dosya ile belirtilmesi ön tanımlı VS.NET davranışı olmasına rağmen, farklı kaynak kod dosyaları içerisinde özniteliklerin belirtilmesi ile de aynı işlev sağlanabilir.

Aşağıda örnek kullanım gösterilmiştir.

**using** System.Reflection;

**using** System.Runtime.CompilerServices;

**using** System.Runtime.InteropServices;

[assembly: AssemblyTitle("USB Port Kütüphanesi")]

[assembly: AssemblyDescription("USB ile ilgili rutinler kütüphanesidir")]

[assembly: AssemblyConfiguration("ÖzelDerleme")]

[assembly: AssemblyCompany("DotNetTurk")]

[assembly: AssemblyProduct("USB Kütüphanesi")]

[assembly: AssemblyCopyright("© DotNetTurk")]

[assembly: AssemblyTrademark("")]

[assembly: AssemblyCulture("Tr-tr")]

[assembly: ComVisible(false)]

[assembly: Guid("15ea6dfd-4b92-469a-b239-5d1486e42ced")]

[assembly: AssemblyVersion("1.0.0.0")]

[assembly: AssemblyFileVersion("1.0.0.0")]

Sürüm Yönetimi

Sürüm yönetimi amacıyla kullanılabilecek öznitelikler ve açıklamalar aşağıda gösterilmiştir.

| Öznitelik | Açıklama |
| --- | --- |
| AssemblyVersion | Assembly'ye ait sürüm bilgisini belirtir özniteliktir. |
| AssemblyFileVersion | Assembly dosyasına (.exe, .dll) ait sürüm bilgisini belirtir özniteliktir. |
| AssemblyInformationalVersion | Ticari sürüm bilgisini belirtir özniteliktir. |

AssemblyVersion ve AssemblyFileVersion öznitelikleri birbirlerinden farklıdır. AssemblyVersion öz niteliği ile amaçlanan, mantıksal düzeyde, GAC ve referans içerisinde bulunan sürüm bilgisini belirtmektir. AssemblyFileVersion öz niteliği, standart Win32 DLL dosyalarında olduğu gibi, dosyanın sürümünü belirtmek amacıyla kullanılmalıdır. Bu öznitelik ile belirtilen sürüm bilgisi Windows gezgini dosya özellikleri penceresi ile görüntülenebilir ve CLR tarafından göz ardı edilir.

Sürüm bilgileri \* kullanımını desteklemektedir. Bu durumda VS.NET tarafından yapılan her derlemede sürüm otomatik olarak artırılacaktır.

Aşağıda sürüm yönetimi ile ilgili örnekler gösterilmiştir.

[assembly: AssemblyVersion("1.0.0.0")]

[assembly: AssemblyFileVersion("1.0.0.0")]

[assembly: AssemblyInformationalVersion("USB Kütüphanesi SP2")]  
  
// Otomatik Sürüm Artırım Örnekleri:  
[assembly: AssemblyVersion("1.0.\*)]

[assembly: AssemblyFileVersion("1.\*)]

Özellikle yazılımın sık değiştiği durumlarda, çok sayıda Assembly sürümünün bulunması ve güçlü isimlendirilmiş Assembly kullanımı dağıtım ve yazılım güncelleme problemlerini de beraberinde getirecektir. Genel prensip olarak, Assembly sürüm numarasını sadece üst sürüm yazılımın alt sürümü desteklemediği durumlarda değiştiriniz. Buna rağmen, kullanıcılara yazılımın her yeni güncellemesini gönderdiğinizde mutlaka Assembly dosya sürümünü artırınız.

Assembly sürümleri sadece güçlü isimlendirilmiş Assembly'ler için anlam ifade etmektedir. Uygulama dağıtımında yöntem olarak XCopy dağıtım ile özel Assembly kullanımı tercih ediliyorsa belirtilen özniteliklerin kullanımına gerek yoktur.

Bir sonraki konuya geçmeden önce şu ana kadar aktarılan bilgilerin kısa bir tekrarı ve bazı kod çalışmaları yapılması önerilmektedir.

Assembly Yükleme

Bu bölümde, şu ana kadar .NET Assembly kullanımı konusunda verilen bilgiler ışığında, bir Assembly tarafından referans verilen bir başka Assembly'nin çalışma zamanında CLR tarafından yüklenmesi hakkında bilgiler aktarılacaktır.

Assembly yükleme, Assembly Binding olarak ifade edilmektedir.

Güçlü İsimlendirilmiş Assembly Yükleme Adımları

Aşağıda güçlü isimlendirilmiş Assembly yükleme adımları gösterilmiştir.

Şekil 27, Güçlü İsimlendirilmiş Assembly Yükleme Adımları

Güçlü isimlendirilmiş Assembly'lerin özel Assembly yöntemi ile kullanımı ve dağıtımı (XCopy Deployment) mümkündür. Böylece programcı, GAC'a yükleme yapmadan güçlü isimlendirme avantajlarını sağlayabilmektedir.

Güçlü İsimlendirilmemiş Assembly Yükleme Adımları

Aşağıda güçlü isimlendirilmemiş Assembly yükleme adımları gösterilmiştir.

Şekil 28, Güçlü İsimlendirilmemiş Assembly Yükleme Adımları

Assembly Yükleme İşlemlerinin İzlenmesi

.NET Framework SDK ile dağıtılan Assembly Binding Log Viewer aracı yardımıyla Assembly yükleme işlemlerinin izlenmesi mümkündür.

> Fuslogvw.exe

Aşağıda araç ekran çıktısı gösterilmiştir.

Ekran Çıktısı 40, .NET Assembly Binding Log Viewer

Assembly Kaynakları

Assembly içerisine her türlü kaynağın gömülmesi ve çalışma zamanında gömülü kaynağın kullanılabilmesi mümkündür.

Kaynak kullanımının sağladığı avantajlardan birisi, uygulama tarafından kullanılan dosyaların ayrı ayrı dağıtımı yerine, tek Assembly dosyası içerisine gömülü olarak dağıtımının sağlanabilmesidir. Sağlanan bir diğer avantaj, uygulamanın aynı olmasına rağmen, farklı kültürler için farklı ara yüzlere sahip olabilmesidir.

Kültür, .NET terminolojisinde dil ve ülkeyi ifade etmektedir. Örneğin "en-US", İngilizce - Birleşik Devletleri ifade etmektedir.  
  
Specific Culture, dil ve ülke bilgisi olan, Neutral Culture sadece dil bilgisi olan kültürü ifade eden kavramlardır.

Kaynak Tipleri

Assembly içerisine istenilen her türlü kaynağı gömmek mümkündür. Gömülen kaynak, çalışma zamanında bir nesne olarak elde edilmekte ve tip dönüşüm kuralları ile istenilen tipe dönüştürülebilmektedir.

Karakter dizi kaynağı (ing. String Resource), bir belirteç yardımıyla istenilen karakter dizi değerini çalışma zamanında elde etmek için kullanılmaktadır. Resim, ses, video veya herhangi bir metin veya ikil dosya gibi her türlü kaynak Assembly içerisine gömülebilmektedir. Kaynağın tipi ile ifade olunan, Assembly içerisinde depolanan bilginin çalışma zamanında elde edilmesi ve farklı tiplere dönüşümünün yapılması ile elde edilen nesnelerdir.

Kaynak Dosya Tipleri

Assembly içerisine gömülmek istenen kaynakları belirtmek için üç tip kaynak dosyası bulunmaktadır. Bunlar metin dosyaları, XML yapısında olan ResX dosyaları ve ikil yapıda olan kaynak dosyalarıdır (Resource Files).

Şekil 29, Kaynak Dosyaları ve Assembly

Metin Kaynak Dosyaları

Metin dosyalarının amacı metin dizi kaynaklarının belirtilmesidir. Aşağıda örnek metin dosya içeriği gösterilmiştir.

Dil=Türkçe  
FormBaslik=Türkçe Başlık  
IleriBtn=İleri  
GeriBtn=Geri

Metin kaynak dosyaları ad – değer içeriğine sahiptir. Ad, kaynak değerini (karakter dizi değerini) elde etmek için kullanılan belirteçtir. Yukarıdaki örnekte, uygulamanın form başlığı ve ileri, geri butonlarının görsel etiketleri belirtilmektedir.

Metin dosyalarının Assembly içerisine gömülebilmeleri için kaynak dosyalarına dönüştürülmesi gerekmektedir. Bu dönüşüm .NET Framework SDK ile dağıtılan kaynak dosya üreticisi (ing. Resource File Generator) aracı ile yapılabilir.

resgen.exe Turkce.txt Turkce.Resources

XML Kaynak Dosyaları

XML kaynak dosyaları (ing. ResX Files), XML yapısında olan ve içerisinde her türlü kaynağın saklanabileceği dosyalardır.

XML kaynak dosyalarının direkt Assembly içerisine gömülebilmeleri mümkün değildir. Metin dosyalarda olduğu gibi bu dosyaların da Resgen.exe ile kaynak dosyasına dönüşümü yapılmalıdır.

Resgen.exe Kaynaklar.resx Kaynaklar.Resources

Kaynak Dosyaları

Kaynak dosyaları ikil yapıda olan ve derleyici tarafından Assembly içerisine gömülebilen dosyalardır.

csc.exe   
 /t:exe   
 /resource: kaynaklar.resources   
 program.cs

Bir Assembly içerisine istenen sayıda kaynak dosyasının gömülmesi mümkündür. Assembly içerisinde, gömülen kaynak dosyasına adı ile erişilmektedir. Gömülü kaynak dosyası adı, çalışma zamanında kaynak adı (ing. Resource Name) olarak ifade edilmektedir (örneğin kaynaklar, resimler, ingilizce).  
  
Kaynak isimleri büyük – küçük harf duyarlıdır. Kaynak isimlendirme standardı olarak küçük harf kullanımı önerilmektedir.

Kaynak Adları

Kaynak adları aşağıdaki yapıdadır.

[Kaynak Adı].[Kültür].resource

Örneğin

kaynaklar.en-US.resource

kaynaklar.de-DE.resource

Yukarıdaki örnekte kaynaklar, kaynağın temel adıdır (ing. Base Resource Name). Kültür adının kullanım amacı, kaynak adı aynı olmasına rağmen kaynak içeriğinin farklı dil ve ülkeler için farklı belirlenebilmesini sağlamaktır. Böylece, uygulama aynı kaynağı kullansa dahi, lokalizasyon (ing. Localization) ile farklı kültürler için çalışma zamanında farklı kaynak içeriklerine erişim mümkün olmaktadır.

Lokalizasyon konusunda kitabın ilerleyen bölümlerinde açıklama yapılacaktır.

System.Resources Ad Alanı

System.Resources ad alanı kaynak yönetimi ile ilgili sınıfların bulunduğu ad alanıdır. Bu bölümde pratikte gerekebilecek sınıflar ile ilgili açıklamalar yapılacaktır.

Kaynak Okuma ve Yazma Arabirimleri

IResourceWriter ve IResourceReader arabirimleri sırasıyla kaynak yazma ve okuma işlemleri için tanımlı arabirimlerdir. Farklı kaynak tiplerinin yönetimini yapan sınıflar bu arabirimlerin gerçekleştirimini yapmaktadır.

IResourceWriter.AddResource metodu kaynak eklemek için, IResourceWriter.Generate metodu eklenen kaynaklardan içeriğin oluşturulması için kullanılmalıdır.

Her iki arabirim de IDisposable arabirimini miras almaktadır. Kullanımda using bloklarının tercih edilmesi önerilmektedir.

XML Kaynak Dosyası Oluşturma ve Okuma

ResXResourceWriter sınıfı XML kaynak dosyası oluşturma ve içeriğini okumak amacıyla kullanılmalıdır.

C#

**using** (ResXResourceWriter writer = **new** ResXResourceWriter(@"c:\kaynaklar.resx")) {

writer.AddResource("IleriBtn", "İleri");

Image img = Image.FromFile(@"c:\ornek.jpg");

writer.AddResource("Resim", img);

writer.Generate();

writer.Close();

}

VB.NET

**Using** writer **As** ResXResourceWriter = **New** ResXResourceWriter("c:\kaynaklar.resx")

writer.AddResource("IleriBtn", "İleri")

**Dim** img **As** Image = Image.FromFile("c:\ornek.jpg")

writer.AddResource("Resim", img)

writer.Generate()

writer.Close()

**End** **Using**

Yukarıdaki örneklerde, içerisinde karakter dizi ve resim yapısında iki adet kaynağı içeren XML kaynak dosyası oluşturulmaktadır. Oluşturulan XML kaynak dosyası kaynaklar.resx ismi ile saklanmaktadır.

XML kaynak dosyasının bir bölümü aşağıda gösterilmiştir.

<?xml version="1.0" encoding="utf-8"?>

<root>  
 . . .  
 <data name="IleriBtn" xml:space="preserve">

<value>İleri</value>

</data>

<assembly alias="System.Drawing"   
 name="System.Drawing, Version=2.0.0.0,   
 Culture=neutral,   
 PublicKeyToken=b03f5f7f11d50a3a" />

<data name="Resim"   
 type="System.Drawing.Bitmap, System.Drawing"   
 mimetype="application/x-...">

<value>

9j/4YpFRXhpZgAASUkqAAgAB  
. . .

ResXResourceReader sınıfı, XML kaynak dosya içeriğinin okunması için kullanılmaktadır. Aşağıda ilgili kaynak kodlar listelenmiştir.

C#

**using** (ResXResourceReader reader = **new** ResXResourceReader(@"c:\kaynaklar.resx")) {

**foreach** (DictionaryEntry entry **in** reader) {

Console.WriteLine(**string**.Format("{0}-{1}", entry.Key, entry.Value));

}

reader.Close();

}

VB.NET

**Using** reader **As** ResXResourceReader = **New** ResXResourceReader("c:\kaynaklar.resx")

**For** Each entry **As** DictionaryEntry **In** reader

Console.WriteLine(**String**.Format("{0}-{1}", entry.Key, entry.Value))

**Next**

reader.Close()

**End** **Using**

ResXResourceReader sınıfı ile açılan kaynaklar.resx isimli dosya içerisinde bulunan kaynaklar listelenmektedir. Aşağıda kod çıktısı gösterilmiştir.

IleriBtn-İleri  
Resim-System.Drawing.Bitmap

Resmi elde etmek için Bitmap sınıf dönüşümü yapılması yeterlidir.   
  
Örneğin;  
  
Bitmap resim = (Bitmap)entry.Value;  
  
Örnekte kullanılan IleriBtn ve Resim kaynak belirteçleridir (ing. Resource Identifier).

Kaynak Dosyası Oluşturma ve Okuma

ResourceWriter sınıfı kaynak dosyası oluşturma ve içeriğini okumak amacıyla kullanılmalıdır.

C#

**using** (ResourceWriter writer = **new** ResourceWriter(@"c:\kaynaklar.resource")) {

writer.AddResource("IleriBtn", "İleri");

Image img = Image.FromFile(@"c:\ornek.jpg");

writer.AddResource("Resim", img);

writer.Generate();

writer.Close();

}

VB.NET

**Using** writer **As** ResourceWriter = **New** ResourceWriter("c:\kaynaklar.resx")

writer.AddResource("IleriBtn", "İleri")

**Dim** img **As** Image = Image.FromFile("c:\ornek.jpg")

writer.AddResource("Resim", img)

writer.Generate()

writer.Close()

**End** **Using**

Yukarıdaki örneklerde, içerisinde karakter dizi ve resim yapısında iki adet kaynağı içeren kaynak dosya oluşturulmaktadır. Oluşturulan kaynak dosyası kaynaklar.resource ismi ile saklanmaktadır.

ResourceReader sınıfı, kaynak dosya içeriğinin okunması için kullanılmaktadır. Aşağıda ilgili kaynak kodlar listelenmiştir.

C#

**using** (ResourceReader reader = **new** ResourceReader(@"c:\kaynaklar.resx")) {

**foreach** (DictionaryEntry entry **in** reader) {

Console.WriteLine(**string**.Format("{0}-{1}", entry.Key, entry.Value));

}

reader.Close();

}

VB.NET

**Using** reader **As** ResourceReader = **New** ResourceReader("c:\kaynaklar.resx")

**For** Each entry **As** DictionaryEntry **In** reader

Console.WriteLine(**String**.Format("{0}-{1}", entry.Key, entry.Value))

**Next**

reader.Close()

**End** **Using**

ResXResourceReader sınıfı ile açılan kaynaklar.resource isimli dosya içerisinde bulunan kaynaklar listelenmektedir. Aşağıda kod çıktısı gösterilmiştir.

IleriBtn-İleri  
Resim-System.Drawing.Bitmap

Assembly Kaynaklarına Erişim

ResourceManager sınıfı, Assembly içerisine gömülü kaynaklara erişim için kullanılan sınıftır. Sınıf, çalışma zamanında kaynak yönetiminin yapılmasını sağlayan çeşitli özellik ve metotlara sahiptir.

Aşağıda temel üye bilgileri gösterilmiştir.

| Üye Adı | Açıklama |
| --- | --- |
| BaseName | Kaynağın temel adını belirtir salt okunur özelliktir. |
| IgnoreCase | GetString ve GetObject metotlarına iletilen kaynak belirteç adında, büyük – küçük harf duyarlılığının kullanıp kullanılmayacağını belirtir. Özelliğe True değerinin atanması büyük – küçük harf duyarlılığını sağlamaktadır. |
| GetString | Kaynak içerisinden karakter dizi değerini elde etmek için kullanılmalıdır. |
| GetObject | Kaynak içerisinde bulunan herhangi bir veriye erişim için kullanılmalıdır. |

Tablo 49, System.Resources.ResourceManager Sınıfı Üye Bilgileri

ResourceManager sınıfı kullanımı ile ilgili çeşitli kaynak kod örnekleri ve açıklamalar aşağıda gösterilmiştir.

C#

**using** System.Resources;

**using** System.Reflection;

. . .

ResourceManager r;

// ResourceManager nesnesini oluştur. "kaynaklar", temel kaynak adını

// ifade etmektedir. Assembly.GetExecutingAssembly metodu aktif Assembly   
// nesnesini belirtmektedir.

r = **new** ResourceManager("kaynaklar", Assembly.GetExecutingAssembly());

// GetString metodu ile karakter dizi kaynak tipine sahip değeri al.

**string** s = r.GetString("IleriBtn");

// GetObject metodu ile her tür gömülü kaynağa erişim mümkündür.

// Elde edilen nesne, tip dönüşümü ile istenilen tipe dönüştürülebilir.

Image img = (Bitmap)r.GetObject("Resim")

VB.NET

**Imports** System.Resources

**Imports** System.Reflection

**Dim** r **As** ResourceManager

' ResourceManager nesnesini oluştur. "kaynaklar", temel kaynak adını

' ifade etmektedir. Assembly.GetExecutingAssembly metodu aktif Assembly   
' nesnesini belirtmektedir.

r = **New** ResourceManager("kaynaklar", [Assembly].GetExecutingAssembly())

**Dim** s **As** **String**

' GetString metodu ile karakter dizi kaynak tipine sahip değeri al.

s = r.GetString("IleriBtn")

' GetObject metodu ile her tür gömülü kaynağa erişim mümkündür.

' Elde edilen nesne, tip dönüşümü ile istenilen tipe dönüştürülebilir.

**Dim** img **As** Image = **CType**(r.GetObject("Resim"), Bitmap)

Kod 135, Kaynak Kullanım Örneği

Kaynak adının kültür bilgisini de içermesi durumunda yapıcı metoda sadece kaynak temel adının iletilmesi yeterlidir. Örneğin kaynak adı kaynaklar.en-US ise, yapıcı metoda kaynaklar değerinin iletimi yeterlidir.   
  
Assembly içerisinde aynı ada sahip ancak farklı kültürler için oluşturulmuş kaynakların (örneğin kaynaklar.tr-TR, kaynaklar.en-US, kaynaklar.de-DE gibi) bulunması durumunda ResourceManager aktif kültür bilgisine göre uygun kaynak içeriğine erişecektir.

Lokalizasyon ve Uydu Assembly Kullanımı

Uydu Assembly (ing. Satellite Assembly), kod gerçekleştirimi olmayan, içerisinde sadece farklı kültürler için kaynakların bulunduğu Assembly'lere verilen isimdir.

Lokalizasyon işleminde kullanılan uydu Assembly'ler, kültür bilgisi içeren diğer Assembly kaynakları gibi, ResourceManager tarafından aktif kullanıcı ara yüz kültür bilgisine (System.Threading.Thread.CurrentUICulture tarafından döndürülen değer) göre yüklenmektedir.

Hazırlık İşlemleri

Konunun anlatımında Türkçe ve İngilizce karakter dizi değerleri içeren kaynak dosyalar kullanılacaktır.

| Kaynaklar.tr-TR.txt | Kaynaklar.en-US.txt |
| --- | --- |
| Dil=Türkçe  Baslik=Uygulamam  Onay=Emin misiniz ? | Dil=English  Baslik=My Application  Onay=Are you sure ? |

Resgen.exe ile kaynaklar.tr-TR.resource ve kaynaklar.en-US.resource isimlerinde kaynak dosyalar oluşturulmuştur.

resgen.exe Kaynaklar.tr-TR.txt  
resgen.exe Kaynaklar.en-US.txt

oluşturulan kaynak dosyaları tr-TR ve en-EN isimli alt klasörlere kopyalanmıştır.

<DIR> en-US

85 Kaynaklar.en-US.txt

76 Kaynaklar.tr-TR.txt

<DIR> tr-TR

Uydu Assembly Oluşturma

Uydu Assembly'ler .NET Framework SDK ile dağıtılan Assembly Linker (al.exe) aracı yardımıyla oluşturulabilir.

en-US> al.exe /embed:kaynaklar.en-US.resources   
 /c:en-US   
 /out:kaynaklar.resources.dll  
  
tr-TR> al.exe /embed:kaynaklar.tr-TR.resources   
 /c:tr-TR   
 /out:kaynaklar.resources.dll

Yukarıdaki örnekte, kaynaklar.en-US.resources isimli kaynak, en-US kültürü ile kaynaklar.resources.dll isimli uydu Assembly'si olarak derlenmektedir. Aynı işlem kaynaklar.tr-TR.resources dosyası için tekrarlanmaktadır.

Lokalizasyon

Uygulama içerisinden uydu Assembly'lerin kullanımı için gerekli adımlar aşağıda gösterilmiştir.

1. Uygulamanın çıktı Assembly adı ile uydu Assembly isimleri aynı olmalıdır. Örneğin kaynaklar.exe.
2. Uygulama klasörü içerisine uydu Assembly'lerinin saklanacağı alt klasörler açılmalıdır. Klasör isimleri kültür bilgisi ile aynı olmalıdır. Örneğin en-US ve tr-TR.

Aşağıda lokalizasyonun örneklendiği kaynak kod ve açıklamalar bulunmaktadır.

C#

**using** System;

**using** System.Collections.Generic;

**using** System.Text;

**using** System.Resources;

**using** System.Reflection;

**using** System.Threading;

**using** System.Globalization;

**class** Program {

**static** **void** Main(**string**[] args) {

ResourceManager r = **new** ResourceManager("kaynaklar",

Assembly.GetExecutingAssembly());

// Aktif kullanıcı ara yüz kültür bilgisini yazdır.

Console.WriteLine("Aktif Kültür: {0}",   
 Thread.CurrentThread.CurrentUICulture.ToString());

// Onay isimli karakter dizi değerini yazdır.

Console.WriteLine(r.GetString("Onay"));

// Yeni bir kültür bilgisi nesnesi oluştur.

CultureInfo trCulture = **new** CultureInfo("tr-TR");

// Aktif kültürü değiştir ve yazdır.

Thread.CurrentThread.CurrentUICulture = trCulture;

Console.WriteLine("Aktif Kültür: {0}",   
 Thread.CurrentThread.CurrentUICulture.ToString());

// GetString metodunu tekrar çağır.

Console.WriteLine(r.GetString("Onay"));

CultureInfo enCulture = **new** CultureInfo("en-US");

// GetString metoduna kültür bilgisinin de iletilmesi ile

// farklı kültürler için geçerli içeriğin elde edilmesi

// mümkün olmaktadır.

Console.WriteLine("Türkçe:{0}", r.GetString("Onay", trCulture));

Console.WriteLine("İngilizce:{0}", r.GetString("Onay", enCulture));

}

}

VB.NET

**Imports** System

**Imports** System.Collections.Generic

**Imports** System.Text

**Imports** System.Resources

**Imports** System.Reflection

**Imports** System.Threading

**Imports** System.Globalization

**Module** Program

**Sub** Main(**ByVal** args() **As** **String**)

**Dim** r **As** ResourceManager = **New** ResourceManager("kaynaklar", \_  
 Assembly.GetExecutingAssembly())

' Aktif kullanıcı ara yüz kültür bilgisini yazdır.

Console.WriteLine("Aktif Kültür: {0}", \_  
 Thread.CurrentThread.CurrentUICulture.ToString())

' Onay isimli karakter dizi değerini yazdır.

Console.WriteLine(r.GetString("Onay"))

' Yeni bir kültür bilgisi nesnesi oluştur.

**Dim** trCulture **As** CultureInfo = **New** CultureInfo("tr-TR")

' Aktif kültürü değiştir ve yazdır.

Thread.CurrentThread.CurrentUICulture = trCulture

Console.WriteLine("Aktif Kültür: {0}", \_  
 Thread.CurrentThread.CurrentUICulture.ToString())

' GetString metodunu tekrar çağır.

Console.WriteLine(r.GetString("Onay"))

**Dim** enCulture **As** CultureInfo = **New** CultureInfo("en-US")

' GetString metoduna kültür bilgisinin de iletilmesi ile farklı kültürler   
 ' için geçerli içeriğin elde edilmesi mümkün olmaktadır.

Console.WriteLine("Türkçe:{0}", r.GetString("Onay", trCulture))

Console.WriteLine("İngilizce:{0}", r.GetString("Onay", enCulture))

**End** **Sub**

**End** **Module**

Kod 136, Lokalizasyon Örneği

Örneğin, aktif ara yüz kültür bilgisi en-US olan bir bilgisayarda çalıştırılması sonucu üretilen çıktı aşağıda gösterilmiştir.

Aktif Kültür: en-US

Are you sure ?

Aktif Kültür: tr-TR

Emin misiniz ?

Türkçe: Emin misiniz ?

İngilizce: Are you sure ?

ResourceManager.GetString metodunun, aktif kültür bilgisine göre erişeceği kaynağı otomatik olarak belirlediğine dikkat edilmelidir.

Listelenen kaynak kodlar \Bölüm 9\Localization klasöründe bulunmaktadır.

VS.NET ile Kaynak Kullanımı

VS.NET, kaynak kullanımını oldukça kolaylaştıran özellikler içermektedir. Bu bölümde, VS.NET tarafından sunulan özellikler yardımıyla kaynak kullanımı hakkında bilgiler verilecektir.

Kaynak Yöneticisi

Proje özellikleri | Resources sekmesi yardımıyla herhangi bir projeye (Konsol uygulaması, sınıf kütüphanesi, vb.) istenilen kaynağın eklenmesi mümkündür.

Ekran Çıktısı 41, VS.NET Kaynak Yöneticisi

Yukarıdaki sayfa yardımıyla projeye eklenen kaynaklar XML kaynak dosyası yapısında, Resources.resx isimli dosyada saklanmaktadır.

Ekran Çıktısı 42, Çözüm Gezgini ve XML Kaynak Dosyası

Dosyanın görüntülenebilmesi için VB.NET projelerinde Show All Files seçeneğinin işaretli olması gerekmektedir.

Projenin derlenmesi aşamasında, XML yapısı VS.NET tarafından kaynak dosya yapısına (.resource) dönüştürülmekte ve Assembly içerisine gömülmektedir.

Resources.Designer.cs ve Resources.Designer.vb (sırasıyla C# ve VB.NET), VS.NET tarafından kaynak yönetimi için otomatik olarak oluşturulan ve gerekli sınıf tanımlarını içeren dosyalardır. Kaynak yöneticisi ile yeni bir kaynak eklenmesi durumunda, bu dosya içerisinde tanımlı Resources sınıfına, gerekli özellikler otomatik olarak eklenmektedir. Böylece çalışma zamanında System.Resources.ResourceManager sınıfı ile yapılabilecek kaynak erişimi Resources sınıfına ait statik özellikler yardımıyla daha kolay yapılabilmektedir.

Örneğin, FormBaslik isminde karakter dizi değerinin kaynak yöneticisi ile proje kaynaklarına eklenmesi durumunda, Resources sınıfına aynı isimde statik özellik VS.NET tarafından otomatik olarak eklenecektir.

Ekran Çıktısı 43, Kaynak Yöneticisi ile Karakter Dizi Değer Ekleme

İşlemden sonra Resources sınıfına eklenen FormBaslik isimli özellik aşağıda gösterilmiştir.

C#

**class** Resources {

. . .

**internal** **static** **string** FormBaslik {

**get** {

**return** ResourceManager.GetString("FormBaslik", resourceCulture);

}

}

. . .

}

VB.NET

**Module** Resources

. . .

**Friend** **ReadOnly** **Property** FormBaslik() **As** **String**

**Get**

**Return** ResourceManager.GetString("FormBaslik", resourceCulture)

**End** **Get**

**End** **Property**

. . .

Özellik kod gerçekleştiriminde System.Resources.ResourceManager.GetString metodunun kullanıldığına dikkat ediniz.

Uygulama içerisinde Resources.FormBaslik özelliği ile kaynak değerinin alınması mümkündür.

VS.NET tarafından sağlanan, kaynak yönetiminin görsel olarak yapılabilmesi ve otomatik kod üretimidir. Daha önceki konularda aktarılan teorik bilgiler VS.NET tarafından kullanılmaktadır.

Kaynakların Gömülmesi

Kaynak yöneticisi ile her tür kaynağın eklenmesi ve otomatik kod üretimi ile kaynağa statik özellikler yardımıyla erişim mümkündür.

Add Resource | Add Existing File ... komutu ile var olan herhangi dosya eklenebilir. Benzer şekilde Add Resource | Add Image ... komutu ile herhangi resim dosyasının eklenmesi mümkündür.

Eklenen kaynaklar, proje dizini içerisinde açılan Resources isimli klasöre kopyalanacak ve XML yapısında olan kaynak dosyası içerisinde ilgili dosyalara referans verilecektir.

XML yapısında olan kaynak dosyası üzerinde sağ tıklanarak Open With ... komutu verilebilir. Bu durumda ön tanımlı editör yerine farklı editörler ile içeriğin incelenmesi mümkündür. Örneğin dosya XML editör ile açılarak XML yapısı görüntülenebilir.

Görsel olarak eklenen her kaynak için Resources sınıfına eklenen özellik tipleri, eklenen kaynağın tipine göre belirlenmektedir.

Örneğin karakter dizi değeri veya metin dosyası eklenmesi durumunda string, ikil dosya eklenmesi durumunda byte [], resim dosyası eklenmesi durumunda Bitmap tipi otomatik olarak kullanılmaktadır.

Kaynak içeriğinin XML kaynak dosya içerisine gömülmesi veya bu dosya içerisinde sadece kaynağa referans verilmesi, kaynağın Persistence özelliği ile yönetilmektedir.

Ekran Çıktısı 44, Kaynak Persistence Özelliği

Ön tanımlı değer olan Linked at Compile Time, kaynak dosyasına ResX dosyası içerisinde referans verilmesi gerektiğini, Embedded in .resx ise kaynak içeriğinin ResX dosyasına gömülmesi gerektiğini belirtmektedir.

Benzer şekilde, kaynak olarak eklenen dosyalar için FileType özelliği, dosya içeriğinin metin bazlı veya ikil olup olmadığını belirtmektedir.

Ekran Çıktısı 45, Kaynak Dosya Tipinin Belirtilmesi

FileType değerinin Text olarak belirtilmesi, özellik tipi için string, Binary olarak belirtilmesi ise özellik tipi olarak byte [] kullanımını sağlamaktadır.

Dosyaların Kaynak Olarak Gömülmesi

VS.NET projesine eklenen herhangi bir dosyanın, kaynak yöneticisini kullanmadan, Assembly içerisine kaynak olarak gömülmesi mümkündür. Bunun için ilgili dosyayı seçip Build Action özelliğinin Embedded Resource olarak belirtilmesi yeterlidir.

Ekran Çıktısı 46, Dosyanın Kaynak Olarak Gömülmesi

Yukarıdaki ekran çıktısında, bilgiler.data isimli dosya projeye Add Existing File... komutu ile eklenmiş ve Build Action özelliği Embedded Resource olarak belirtilmiştir. Bu durumda dosya içeriği, projenin derlenmesi aşamasında Assembly içerisine gömülecektir.

Dosya özelliklerinden Copy to Output Directory seçeneğinin True olarak belirtilmesi, ilgili dosyanın proje çıktı klasörüne (örneğin Debug\Bin) otomatik olarak kopyalanmasını sağlamaktadır.

Aşağıda, çalışma zamanında kaynak erişimi gösterilmiştir.

C#

Assembly a;

a = Assembly.GetExecutingAssembly();

Stream stream;

stream = a.GetManifestResourceStream("ConsoleApplication4.Bilgiler.data");

StreamReader reader;

reader = **new** StreamReader(stream);

**string** content = reader.ReadToEnd();

Console.WriteLine(content)

VB.NET

**Dim** a **As** Assembly

a = Assembly.GetExecutingAssembly()

**Dim** stream **As** Stream

stream = a.GetManifestResourceStream("ConsoleApplication4.Bilgiler.data")

**Dim** reader **As** StreamReader

reader = **New** StreamReader(stream)

**Dim** content **As** **String** = reader.ReadToEnd()

Console.WriteLine(content)

Assembly.GetManifestResourceInfo ve Assembly.GetManifestResourceNames metotları Assembly içerisine gömülü kaynaklar hakkında bilgi almak ve kaynak isimlerini listelemek amacıyla kullanılabilir.  
  
Stream kullanımı konusunda kitabın ilerleyen bölümlerinde detaylı açıklama yapılacaktır.

Dinamik Assembly Kullanımı

Derleyici komut satırı parametresi (/Reference:) veya VS.NET ile herhangi bir Assembly dosyasına referans verildiğinde, verilen referans statik olarak adlandırılmaktadır. Statik referanslar, Assembly'nin yüklenmesi aşamasında çözümlenmekte ve referans verilen Assembly'ler CLR tarafından yüklenmektedir.

Dinamik Assembly kullanımı ile, çalışma zamanında Assembly dosyasının yüklenmesi, yüklenen Assembly hakkında çeşitli bilgilere erişim ve yansıma ile Assembly içerisindeki tip tanımlarına erişim mümkündür.

Dinamik Assembly kullanımının arabirimler ile birleştirilmesi, genişleyebilir yazılımlar elde etmek için kullanılabilmektedir. Plugin kullanımı olarak da ifade edilebilen bu yöntem ile, uygulama yazılımı tek olmasına rağmen, belirlenen arabirim kod gerçekleştirimini yapan sınıflar içeren Assembly'ler sisteme yüklenebilmekte ve dinamik olarak kullanılabilmektedir. Plugin kullanımı ve dinamik Assembly tiplerine erişim konusu kitabın Yansıma ve Öznitelikler bölümünde aktarılmıştır.

Assembly Sınıfı

System.Reflection.Assembly sınıfı, Assembly'lerin çalışma zamanında yönetiminin yapılmasını sağlayan sınıftır.

Aşağıda temel sınıf üyeleri ve açıklamaları gösterilmiştir.

| Üye Adı | Açıklama |
| --- | --- |
| FullName | Assembly bütünleşik adını belirtir özelliktir. |
| GlobalAssemblyCache | Assembly'nin GAC içerisinden yüklenmesi durumunda True, özel Assembly olması durumunda False değerinin döndürür özelliktir. |
| Location | Assembly'nin yüklendiği klasörü ve Assembly dosya adını (yolu) belirtir özelliktir. |
| EntryPoint | Assembly'nin giriş metodunu MethodInfo nesnesi olarak döndürür özelliktir. Sınıf kütüphanelerin bu özellik null (VB.NET için nothing) değerini döndürmektedir. Uygulamalar için bu özellik Main metodunu döndürmektedir. |
| GetName | Assembly adını AssemblyName nesnesi olarak döndüren metottur. |
| GetExecutingAssembly | Aktif çalışan Assembly nesnesini elde etmek için kullanılması gereken statik metottur. |
| GetCallingAssembly | Aktif Assembly'yi çağıran Assembly nesnesini elde etmek için kullanılması gereken statik metottur. |
| GetEntryAssembly | Uygulamayı çalıştıran Assembly nesnesini ifade eden statik metottur. |
| Load | İstenilen Assembly'yi yüklemek için kullanılması gereken statik metottur. |
| LoadFrom | İstenilen Assembly dosyasını belirtilen yol içerisinden yüklemek için kullanılması gereken statik metottur. |
| GetReferencedAssemblies | Assembly tarafından referans verilen Assembly isimlerini AssemblyName nesne dizisi olarak döndüren metottur. |

Tablo 50, System.Reflection.Assembly Sınıfı Üye Bilgileri

AssemblyName sınıfı, Assembly adını ifade eden sınıftır. Bu sınıf yardımıyla Assembly adını karakter dizi olarak kullanmak yerine AssemblyName nesnenin kendisi ile Assembly adının belirtilmesi mümkündür.

C#

AssemblyName aName = **new** AssemblyName();

aName.Name = "DotNetKitabı.Lib";

aName.Version = **new** Version("2.0.0.0");

aName.CultureInfo = **new** CultureInfo("tr-TR");

aName.SetPublicKeyToken(**new** **byte**[] {66, 54, 34, 22, 43, 88, 77, 88 });

Console.WriteLine(aName.FullName);

AssemblyName tName = **new** AssemblyName("Advantage.Data.Provider, Version=8.10.2.0, Culture=neutral, PublicKeyToken=E33137C86A38DC06");

Console.WriteLine("Assembly Adı:{0}", tName.Name);

Console.WriteLine("Majör Sürüm :{0}", tName.Version.Major);

Console.WriteLine("Kültür :{0}", tName.CultureInfo.DisplayName);

// AssembyName nesnesi ile yükle.

Assembly a = Assembly.Load(tName);

VB.NET

**Dim** aName **As** AssemblyName = **New** AssemblyName()

aName.Name = "DotNetKitabı.Lib"

aName.Version = **New** Version("2.0.0.0")

aName.CultureInfo = **New** CultureInfo("tr-TR")

aName.SetPublicKeyToken(**New** **Byte**() {66, 54, 34, 22, 43, 88, 77, 88})

Console.WriteLine(aName.FullName)

**Dim** tName **As** AssemblyName = **New** AssemblyName("Advantage.Data.Provider, Version=8.10.2.0, Culture=neutral, PublicKeyToken=E33137C86A38DC06")

Console.WriteLine("Assembly Adı:{0}", tName.Name)

Console.WriteLine("Majör Sürüm :{0}", tName.Version.Major)

Console.WriteLine("Kültür :{0}", tName.CultureInfo.DisplayName)

' AssembyName nesnesi ile yükle.

**Dim** a **As** Assembly = Assembly.Load(tName)

Kod çıktısı aşağıda gösterilmiştir.

DotNetKitabı.Lib, Version=2.0.0.0, Culture=tr-TR, PublicKeyToken=423622162b584d58

Assembly Adı:Advantage.Data.Provider

Majör Sürüm :8

Kültür :Invariant Language (Invariant Country)

Assembly.GetReferencedAssemblies metodu ile referans verilen Assembly'ler elde edilebilmektedir. Bu metodun, uygulama tarafından kullanılan Assembly'lerin, örneğin, uygulamanın hakkında bölümünde listelenmesi amacıyla kullanımı mümkündür.   
  
Böylece, Assembly.Version özelliği ile de birleştirildiğinde, sistemde hangi Assembly'lerin hangi sürümleri ile birlikte yüklü olduğu belirlenebilir.

Dinamik Assembly Yükleme

Assembly.Load ve Assembly.LoadFrom, çalışma zamanında Assembly dosyalarının dinamik olarak yüklenmesi için kullanılan metotlardır.

Load metoduna sadece Assembly adının parametre olarak iletilebilmektedir (örneğin DotNetKitabı.Lib). Bu durumda, daha önceki konularda aktarılan Assembly yükleme işlemleri yapılmakta ve Assembly yüklenmektedir. LoadFrom metodu ile, Assembly dosyası ve bulunduğu klasör belirtilebilir (örneğin C:\DotnetKitabı.Lib.dll).

Load metoduna, AssemblyName nesnesinin parametre olarak iletimi mümkündür.

Bir insanın bilgisi vardır, pratiği yoktur, fakat bir diğerinin bilgisi de, pratiği de, yeterli tecrübesi de vardır. Bunu şöyle açıklayabiliriz: Bir şahıs bir hastalığın ilacını kitaplarda arar, bulur, okur, üstelik bütün ilaçların isimlerini de bilir, o kadar. Bir diğeri bütün ilaçların isimlerini bildiği gibi, hasta tedavi etmiş ve defalarca tecrübe yapmış ise, bu iki zat asla aynı olamaz.

Nizamülmülk

Bölüm

10

Prosesler ve Uygulama Etki Alanları

Ü

zerinde çalışılan platformun mimarisel düzeyde anlaşılabilmesi ve arka planda yapılan işlemler hakkında bilgi sahibi olmak, .NET üzerinde daha hızlı ve kaliteli uygulama geliştirmek için büyük önem taşımaktadır. Bu bilgiler, aynı zamanda, olası problemlerin daha erken teşhisini ve programcı tarafından çok daha kısa sürede çözülebilmesini sağlamaktadır.

Bu bölümde, Windows prosesleri ve .NET uygulama etki alanları hakkında bilgiler verilecektir.

Windows Prosesleri

Proses (ing. Process) tanımı aşağıdaki şekilde yapılabilir.

Proses, çalışan bir uygulama yazılımı tarafından kullanılan kaynakları belirten kavramdır.

Daha somut olmak gerekirse, Windows platformunda proses, çalışan herhangi bir uygulamayı belirtmek için kullanılan terimdir. Çalışan uygulama, Windows, konsol veya hizmet uygulaması tipinde olabilir.

Sistemde çalışan prosesleri görüntülemek için Windows Task Manager kullanılabilir.

Ekran Çıktısı 47, Windows Task Manager

Windows Task Manager, Ctrl+Shift+Esc tuş kombinasyonu ile açılabilir.  
  
Task Manager içerisinde PID (Process Id) sütunun görüntülenebilmesi için View | Select Columns ... komutu verilmeli ve PID kutusu işaretlenmelidir.

Proses numarası (ing. Process Id), işletim sistemi tarafından prosese verilen ve tekil olan numaradır. Bu numara, sistemde çalışan bir çok proses içerisinde ilgili prosesi tekil olarak ifade etmek amacıyla kullanılmaktadır.

Herhangi bir çalıştırılabilir dosya'nın (.exe) üzerine tıklanarak çalıştırılması durumunda Windows tarafından yeni bir proses oluşturulmakta ve Windows Task Manager | Processes sekmesinde görüntülenmektedir.

Microsoft tarafından ücretsiz olarak sunulan Process Explorer, çalışan prosesler hakkında çok daha detaylı bilgi alabilmeyi sağlamaktadır.

\Yardımcı Araçlar\SysInternals\ klasöründe Process Explorer ve çeşitli sistem araçları bulunmaktadır.

Windows prosesleri ve proseslerin yönetimi .NET platformuna özgü değildir. Kitabın daha önceki konularında da bahsedildiği gibi, .NET platformu, Windows işletim sistemi üzerinde çalışmakta ve uygulamalar için yönetilen (ing. Managed Platform) platform sağlamaktadır.

Windows işletim sistemi altında çalışan her yazılım, Windows tarafından belirlenen kural ve standartlara uymak zorundadır. Bu kurallar .NET uygulamaları için de geçerlidir. Bilindiği gibi, yönetilen platform ve yönetilmeyen platform (ing. Unmanaged Platform) – işletim sistemi platformu - (ing. Unmanaged Platform) arasında CLR bulunmaktadır.

Prosesler ve İş Parçacıkları

Herhangi bir uygulamanın Windows tarafından çalıştırılması aşamasında aşağıdaki adımlar sırsıyla yapılmaktadır.

1. Dosyanın geçerli bir Windows çalıştırılabilir dosyası olduğu kontrol edilmektedir.
2. Uygulama tipi dosya içeriğinden elde edilmektedir.
3. Uygulamanın Windows GUI masaüstü (Windows Forms) uygulaması olması durumunda WinMain metodu, konsol uygulaması olması durumunda Main, hizmet uygulaması olması durumunda ServiceMain metodu çağrılmakta ve uygulama çalışmaya başlamaktadır.

İş parçacıkları (ing. Threads), proses içerisindeki çalışma yolları olarak tanımlanabilir. Her prosesin en az bir iş parçacığı bulunmaktadır ve bu iş parçacığı temel iş parçacığı (ing. Primary Thread) olarak ifade edilmektedir. Temel iş parçacığı, yeni iş parçacıkları oluşturabilmekte, böylece proses içerisinde birden çok iş parçacığı paralel olarak çalışabilmektedir.

Şekil 30, Proses ve İş Parçacıkları

Proses içerisinde tek iş parçacığının bulunması durumunda uygulama tekli iş parçacık uygulaması (ing. Single-Threaded Application), birden çok iş parçacığının bulunması durumunda uygulama, çoklu iş parçacık uygulaması (ing. Multi-Threaded Application) olarak tanımlanmaktadır.

Bir proses içerisinde birden çok iş parçacığının bulunması, uygulamanın birden fazla işlemi aynı anda yapabilmesini sağlamaktadır. Örneğin kullanıcı uygulama ile veri girişi yaparken, aynı uygulama arka planda büyük bir dokümanı yazdırabilmekte veya TCP ile veri alışverişi yapabilmektedir.   
  
İş parçacıkları konusunda geniş açıklama kitabın ilerleyen bölümlerinde yapılacaktır. Bu konuya yabancı olan okuyucuların şimdilik, bir prosesin içerisinde en az bir tane olmak şartıyla, birden fazla iş parçacığının bulunabileceği ve bulunan iş parçacıklarının paralel çalışabileceğini bilmeleri yeterlidir.

.NET Uygulamalarının Yüklenmesi

Bilindiği gibi .NET uygulamaları, standart Windows PE (Windows Portable Executable) yapısındadır. Bu yapı ile, .NET uygulamalarının Windows platformu üzerinde direkt çalıştırılabilmelerini sağlanmaktadır. Buna rağmen, standart bir Windows .exe dosyası ile .NET Assembly yapısında olan .exe dosyası birbirlerinden içerik olarak tamamen farklıdır.

Herhangi bir .NET uygulamasının çalıştırılması aşamasında aşağıdaki işlemler sırasıyla yapılmaktadır.

1. Dosyanın geçerli bir Windows çalıştırılabilir dosyası olduğu (PE yapısında olduğu) Windows tarafından kontrol edilmektedir.
2. Windows tarafından mscoree.dll dosyası yüklenmektedir. İşletim sistemi açısından değerlendirildiğinde, standart Windows DLL yapısında olan bu dosya yüklenmiş ve uygulama çalışmaya başlamıştır.
3. mscoree.dll tarafından yapılan işlem, sistemin bir veya birden fazla işlemci içermesine göre mscorwks.dll veya mscorsvr.dll isimli dosyaları yüklemektir. CLR yüklemesi olarak ifade edilen bu işlem ile kontrol .NET platformuna geçmektedir.
4. Kitabın daha önceki bölümlerinde aktarılan Assembly yükleme işlemi CLR tarafından yapılmakta ve uygulama çalışmaktadır.

Aynı bilgisayar üzerinde farklı .NET Framework sürümleri bulunabilmektedir. Bu durumda sistem kayıt defteri içerisinde bulunan HKEY\_LOCAL\_MACHINE – SOFTWARE – Microsoft - .NETFramework anahtarı içerisinde InstallRoot değeri alınmakta ve belirtilen sürüm kullanılmaktadır.

Uygulamanın sadece istenilen .NET Framework sürümünün karşı sistemde yüklü olması durumunda çalışmasını sağlamak için, uygulama konfigürasyon dosyası içerisinde aşağıda gösterilen tanım bulunmalıdır.

<?xml version="1.0" encoding="utf-8" ?>

<configuration>

<startup>

<requiredRuntime **version** ="**1.1.4322**"/>

</startup>

</configuration>

System.Diagnostics.Process Sınıfı

Process sınıfı, .NET sınıf kütüphanesi içerisinde tanımlı ve proseslerin yönetimi için kullanılan sınıftır. Process sınıfı özellik ve metotları yardımıyla, sistemde çalışan proseslerin listesini veya herhangi bir proses hakkında detaylı bilgi alınabilmektedir. Aynı zamanda, uygulama içerisinden herhangi bir Windows uygulaması Process.Start statik metodu ile çalıştırılabilir.

System.Diagnostics ad alanı içerisinde proses, olay görüntüleyicisi ve performans yönetimi ile ilgili sınıflar bulunmaktadır. Kitabın ilerleyen bölümlerinde bu ad alanı ile ilgili detaylı bilgi sunulacaktır.  
  
Process sınıfının sadece .NET uygulamaları ile ilişkili olmadığına ve tüm Windows proseslerinin bu sınıf ile yönetiminin yapılabileceğine dikkat edilmelidir.

Aşağıda Process sınıfı temel üye bilgileri gösterilmiştir.

| Üye Adı | Açıklama |
| --- | --- |
| Id | Prosese işletim sistemi tarafından verilen tekil numarayı belirtir özelliktir. |
| ProcessName | Proses adını belirtir özelliktir. |
| ExitCode | Proses çıkış kodunu belirtir özelliktir. Bir uygulamanın çıkış kodunun 0 olması uygulamanın başarıyla çalıştığını ve kapandığını, 0'dan farklı olması hata ile karşılaşıldığını ve kapandığını belirtmektedir. Bu değer Main metodunun döndürdüğü değerdir. |
| ExitTime | Prosesin çıkış zamanını belirtir özelliktir. |
| MachineName | Prosesin çalıştığı bilgisayar adını belirtir özelliktir. |
| Handle | Prosese işletim sistemi tarafından atanan tanıtıcı değeri belirtir özelliktir. |
| MainWindowTitle | Proses ana pencere başlığını belirtir özelliktir. |
| PriorityClass | Proses önceliğini (ing. Priority) belirtir özelliktir. |
| Threads | Proses tarafından oluşturulan iş parçacık listesini döndürür özelliktir. |
| Start | Yeni proses oluşturmak için kullanılması gereken statik metottur. |
| Kill | Prosesi durdurmak için kullanılması gereken metottur. |
| GetCurrentProcess | Aktif proses özelliklerine göre yeni Process nesnesi döndürür statik metottur. |
| GetProcessesById | İşletim sisteminde çalışan prosese numarası ile erişim için kullanılması gereken statik metottur. |
| GetProcesses | İşletim sisteminde çalışan proseslerin listesini döndürür statik metottur. |
| CloseMainWindow | Proses ana penceresini kapatmak için kullanılması gereken metottur. |

Tablo 51, System.Diagnostics.Process Sınıfı Üye Bilgileri

Proses metotlarının büyük bölümü gerek aynı bilgisayar, gerek farklı bir bilgisayar üzerinde işlemlerin yapılabilmesini sağlamaktadır. Örneğin, yeterli ayrıcalıkların bulunması durumunda, GetProcesses metoduna bilgisayar adı iletilerek uzak bilgisayarda çalışan proses listesi alınabilir.

Sistem Proseslerinin Listelenmesi

Sistem proseslerinin listelenmesi için statik GetProcesses metodu kullanılmalıdır. Aşağıda çalışan proseslerin numara ve isimlerini listeleyen örnek kod parçası gösterilmiştir.

C#

Process[] list = Process.GetProcesses();

**foreach** (Process p **in** list) {

Console.WriteLine("PID:{0}\tAd:{1}", p.Id, p.ProcessName);

}

VB.NET

**Dim** list() **As** Process = Process.GetProcesses()

**For** **Each** p **As** Process **In** list

Console.WriteLine("PID:{0}" + vbTab + "Ad:{1}", p.Id, p.ProcessName)

**Next**

Kod 137, Sistem Proseslerinin Listelenmesi

Yeni Proses Oluşturma

Yeni bir proses oluşturmak için statik Start metodu kullanılmalıdır. Not defteri uygulamasını açan ve kapatan kod örneği aşağıda gösterilmiştir.

C#

// Not defteri uygulamasını c:\test.txt komut satırı parametresi ile   
// çalıştır.

Process p = Process.Start("notepad.exe", @"C:\test.txt");

Console.ReadLine();

// Not defteri ana penceresini kapat.

p.CloseMainWindow();

VB.NET

' Not defteri uygulamasını c:\test.txt komut satırı parametresi ile çalıştır.

**Dim** p **As** Process = Process.Start("notepad.exe", "C:\test.txt")

Console.ReadLine()

' Not defteri ana penceresini kapat.

p.CloseMainWindow()

Kod 138, Yeni Proses Oluşturma

Proses Hakkında Bilgi Almak

Çalışan bir proses hakkında bilgi almak için, proses numarasının parametre olarak iletildiği GetProcessById statik metodu kullanılabilir. Kod örneği aşağıda gösterilmiştir.

C#

Process p = Process.GetProcessById(1620);

Console.WriteLine("PID:{0}\tAd:{1}", p.Id, p.ProcessName);

VB.NET

**Dim** p **As** Process = Process.GetProcessById(1620)

Console.WriteLine("PID:{0}" + vbTab + "Ad:{1}", p.Id, p.ProcessName)

Kod 139, Proses Hakkında Bilgi Almak

Geçersiz proses numarasının metoda iletilmesi durumunda ArgumentException fırlatılacaktır.

Proses Kapatmak

Çalışan bir prosesi kapatmak için CloseMainWindow ve Kill metotları kullanılabilir.

Windows GUI uygulamalarını kapatmak için CloseMainWindow metodu kullanılmalıdır. Bu metot, uygulamanın ana penceresine kapat mesajı göndermektedir.

Kill metodu, prosesi durdurmak (veya öldürmek) için kullanılmalıdır. Bu metot ana penceresi olmayan veya yanıt vermeyen prosesleri kapatmak için kullanılmalıdır.

Uygulama Etki Alanları

Uygulama etki alanları (ing. Application Domains), aynı proses içerisinde birden çok Assembly'nin birbirlerinden izole edilmiş olarak çalışmalarını sağlayan mantıksal kapsayıcılardır.

Uygulama etki alanları sadece .NET platformuna özgü bir kavramdır. Nasıl ki Windows içerisinde aynı anda birden çok proses birbirlerinden izole edilmiş olarak çalışıyor ise, aynı işlev .NET içerisinde uygulama etki alanları yardımıyla sağlanmaktadır. Bu açıklamadan yola çıkılarak ve proses kavramı ile ifade edilmek isteneni tam olarak bilerek, uygulama etki alanı, .NET içerisindeki proses olarak da değerlendirilebilir.

CLR tarafından uygulamanın çalıştırılması aşamasında varsayılan uygulama etki alanı oluşturulmakta ve Assembly'ler bu uygulama etki alanı içerisine yüklenmektedir. CLR, gerektiği durumlarda yeni uygulama etki alanları oluşturmaktadır.

Uygulama etki alanları ile sağlanan temel kazanımlar aşağıda listelenmiştir.

1. Assembly'lerin aynı proses içerisindeki farklı uygulama alanlarına yüklenmesi ile Assembly'ler birbirlerinden tamamen izole edilmiş olarak çalışmaktadır. Böylece, birbirlerinin hafıza alanları ve kaynaklarına erişim engellenmektedir. Bu düzeyde ayrımın CLR tarafından yapılıyor olması güvenlik yönüyle oldukça önemli bir kazanımdır.
2. Herhangi bir uygulama etki alanına yüklenen Assembly, ilgili uygulama etki alanı aktif olduğu sürece yüklü kalmaktadır. Özellikle büyük Assembly'ler ile çalışırken bu durum performans açısından negatif etkilere sahip olabilmektedir.   
     
   Uygulama etki alanları yardımıyla, yeni bir uygulama etki alanı oluşturulabilmekte, Assembly bu uygulama etki alanına yüklenebilmektedir. Gerekli işlemler yapılıp uygulama etki alanının kaldırılması (ing. Unload) ile Assembly de kaldırılmış olmaktadır.
3. Varsayılan uygulama etki alanına yüklenen bir Assembly kararsız duruma düştüğünde, proses de bu durumdan etkilenmektedir. Aynı proses içerisinde yeni bir uygulama etki alanı oluşturulması ve Assembly'nin bu etki alanına yüklenmesi ile, kararsızlığın sadece yeni oluşturulan uygulama etki alanı ile sınırlı kalması sağlanabilmekte, böylece proses kararlılığını sürdürebilmektedir.

Aşağıda genel mimari gösterilmiştir.

Şekil 31, Uygulama Etki Alanları ve Genel Mimari

Uygulama etki alanı içerisinde birbirlerinden izole edilmiş olarak yüklenen Assembly'ler bulunmaktadır. Aynı zamanda, her Assembly kendi uygulama etki alanını oluşturabilmekte ve istediği Assembly'leri uygulama etki alanı içerisine yükleyebilmektedir.  
  
Uygulama etki alanı içerisinde bir veya birden fazla kontekst bulunabilmektedir. Kontekstler, iş parçacıklarının çalıştığı mantıksal kapsayıcılar olarak tanımlanabilir. Uygulama etki alanlarında olduğu gibi, varsayılan kontekst otomatik olarak oluşturulmaktadır.

Uygulama etki alanları kullanımının en güzel örneği ASP.NET uygulamalarını çalıştıran Microsoft Internet Information Server - IIS uygulama sunucusudur. ASP.NET web sitesine gelen istekler IIS tarafından ASP.NET işçi proses (ing. Worker Process) adı verilen prosese iletilmektedir. Internet tarayıcısı ile farklı kullanıcılar tarafından web sitesine istekte bulunulması durumunda, işçi proses bağlı her bir kullanıcı için ayrı uygulama etki alanları oluşturmaktadır. Böylece, kullanıcıların birbirlerinin alanlarına erişimleri engellenmiş olmaktadır.

System.AppDomain Sınıfı

AppDomain sınıfı, uygulama etki alanları yönetiminin yapılmasını sağlayan sınıftır. Aşağıda temel sınıf üyeleri ve açıklamaları gösterilmiştir.

| Üye Adı | Açıklama |
| --- | --- |
| Id | Proses içerisinde uygulama etki alanına verilen tekil numarayı belirtmektedir. |
| CurrentDomain | Aktif uygulama etki alan nesnesini döndürür statik özelliktir. |
| BaseDirectory | Assembly yükleme sırasında Assembly dosyasının aranacağı klasörü belirtmektedir. |
| CreateDomain | Yeni uygulama etki alanı oluşturmak için kullanılması gereken statik metottur. |
| GetAssemblies | Uygulama etki alanı içerisine yüklenen Assembly'leri almak için kullanılmalıdır. |
| Load | İstenilen Assembly'yi uygulama etki alanına dinamik olarak yüklemek için kullanılmalıdır. |
| ExecuteAssembly | İstenilen Assembly'yi uygulama etki alanı içerisinde çalıştırmak için kullanılmalıdır. ExecuteAssemblyByName metodu referans verilen Assembly'yi çalıştırmak için kullanılabilir. |
| Unload | İstenilen etki alanını kaldırmak için kullanılması gereken statik metottur. |

Tablo 52, System.AppDomain Sınıfı Üye Bilgileri

Yeni Uygulama Etki Alanı Oluşturma ve Kaldırma

Assembly içerisinden yeni bir uygulama etki alanı oluşturmak için statik CreateDomain, var olan uygulama etki alanını kaldırmak için statik Unload metodu kullanılmalıdır.

Gerekli kod örneği aşağıda gösterilmiştir.

C#

// Yeni bir uygulama etki alanı oluştur.

AppDomain d = AppDomain.CreateDomain("Alan");

// Oluşturulan uygulama etki alanı içerisindeki yüklü Assebly'lerin

// isimlerini yazdır.

**foreach** (Assembly a **in** d.GetAssemblies()) {

Console.WriteLine(a.GetName().Name);

}

// Uygulama etki alanını kaldır.

AppDomain.Unload(d);

VB.NET

' Yeni bir uygulama etki alanı oluştur.

**Dim** d **As** AppDomain = AppDomain.CreateDomain("Alan")

' Oluşturulan uygulama etki alanı içerisindeki yüklü Assebly'lerin

' isimlerini yazdır.

**For** Each a **As** Assembly **In** d.GetAssemblies()

Console.WriteLine(a.GetName.Name)

**Next**

' Uygulama etki alanını kaldır.

AppDomain.Unload(d)

Kod 140, Yeni Uygulama Etki Alanı Oluşturulması

Uygulama Etki Alanına Assembly Yükleme ve Çalıştırma

Uygulama etki alanı içerisine Assembly yüklemek ve çalıştırmak için sırasıyla Load ve ExecuteAssembly metotları kullanılmalıdır.

Gerekli kod örneği aşağıda gösterilmiştir.

| C# | VB.NET |
| --- | --- |
| AppDomain d;  d = AppDomain.CreateDomain("Alan");  Assembly a;  // DotNetKitabı.Lib isimli Assembly // dosyasını yükle. a = d.Load("DotNetKitabı.Lib");  // uygulama.exe isimli uygulamayı // çalıştır. Komut satır parametresi // de kullanılabildiğine dikkat  // ediniz. d.ExecuteAssembly("uygulama.exe /?");  // Uygulamayı direkt adı ile // çalıştır. d.ExecuteAssemblyByName("uygulama"); | **Dim** d **As** AppDomain  d = AppDomain.CreateDomain("Alan")  **Dim** a **As** Assembly  // DotNetKitabı.Lib isimli Assembly // dosyasını yükle. a = d.Load("DotNetKitabı.Lib")  ' uygulama.exe isimli uygulamayı ' çalıştır. Komut satır parametresi ' de kullanılabildiğine dikkat  ' ediniz. d.ExecuteAssembly("uygulama.exe /?")  ' Uygulamayı direkt adı ile ' çalıştır. d.ExecuteAssemblyByName("uygulama") |

Tablo 53, Uygulama Etki Alanına Assembly Yüklenmesi

Bana okuduğum kitapların en güzelinin hangisi olduğunu sorarsanız, söyleyeyim: ANNEM'dir.

Abraham LINCOLN

Bölüm

11

Yansıma ve Öznitelikler

İ

ngilizce karşılıkları ile Reflection ve Attributes, .NET platformunda sık kullanılan işlem ve yapılardır. Özellikle CLR tarafından arka planda yapılan işlemlerin daha iyi anlaşılması için her iki kavram da büyük önem taşımaktadır.

Yansıma

.NET platformunun sağladığı kazanımlardan birisi de çalışma zamanında Assembly'lerin yüklenebilmesi, tip bilgilerinin alınabilmesi ve dinamik olarak nesne oluşturabilmektir.

Yansıma (ing. Reflection), çalışma zamanında tip bilgisinin elde edilebilmesini ifade eden kavramdır. Yansıma, aynı zamanda, kodun model olarak değerlendirilmesi ile programcının çalışma zamanında kod üretebilmesini de sağlamaktadır.

Yansıma ile ilgili sınıflar System.Reflection ad alanı içerisinde bulunmaktadır. Ad alanı içerisinde bulunan temel sınıflar ve açıklamaları aşağıda gösterilmiştir.

| Sınıf | Açıklama |
| --- | --- |
| Assembly | Assembly yükleme ve Assembly hakkında bilgi elde etme amacıyla kullanılan sınıftır. |
| MemberInfo | Herhangi bir tipin üyesini ifade eden soyut sınıftır. |
| PropertyInfo | Özellik bilgilerini içeren sınıftır. |
| MethodInfo | Metot bilgilerini içeren sınıftır. |

Tablo 54, System.Reflection Ad Alanı Sınıfları

Verilen bir tipe ait her türlü bilgiye yansıma ile erişmek mümkündür. .NET içerisinde tip bilgisi de bir nesne olarak ifade edilmektedir. System.Type sınıfı, tip yönetiminin yapılabileceği özellik ve metotlar içeren sınıftır.

Kitabın önceki konularında Type sınıfı üyeleri hakkında çeşitli bilgiler sunulmuştur.

System.Reflection ad alanı içerisinde yansıma ile ilgili bir çok sınıf bulunmaktadır. Bu bölümde tüm sınıfların detayları aktarılmayacak, sadece pratikte gerekebilecek sınıflar ve konular hakkında bilgiler verilecektir.

Tip Bilgisinin Elde Edilmesi

Tip bilgisi aşağıdaki yöntemlerden herhangi biri ile elde edilebilir.

1. System.Object.GetType metodu.
2. Type.GetType metodu.
3. typeof ve GetType (sırasıyla C# ve VB.NET) rezerve kelimeleri.
4. System.Reflection.Assembly sınıfı metotları.

Bu konu başlığında tip isimleri ve listelenen yöntemlerin kullanımı hakkında bilgiler verilecektir.

Tip Adları

Tip adını üç şekilde ifade etmek mümkündür.

Basit ad, Type.Name özelliği ile ifade edilmekte ve tipin tanım adını belirtmektedir.

Bütünleşik ad, ad alanı ile birlikte tip adıdır. Bütünleşik ad Type.FullName özelliği ile ifade edilmektedir.

Assembly bütünleşik adı, bütünleşik tip adı ile birlikte tipin tanımlandığı Assembly adının da bulunduğu değerdir. Bu değer Type.AssemblyQualifiedName özelliği ile belirtilmektedir.

Örneğin tam sayı tipi (System.Int32) için aşağıda sırasıyla basit, bütünleşik ve Assembly bütünleşik adları gösterilmiştir.

Int32

System.Int32

System.Int32, mscorlib, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089

Bir sınıf veya yapı içerisinde tanımlı alt tipler için + karakteri kullanılmaktadır. Örneğin

| C# | VB.NET |
| --- | --- |
| **namespace** Ornekler {  **class** Test {  **public** **enum** Renk {  Siyah,  Beyaz  }  }  } | **NameSpace** Ornekler  **Class** Test  **Public** **Enum** Renk  Siyah  Beyaz  **End** **Enum**  **End** **Class**  **End NameSpace** |

Yukarıdaki örnekte Test isimli sınıf içerisinde Renk isimli sayılabilir tip tanımı yapılmıştır. Renk tipi aşağıdaki şekilde ifade edilmektedir.

Ornekler.Test+Renk

Nesnelerin Tip Bilgilerine Erişim

Nesnelerin tip bilgisini almak için GetType metodu kullanılmaktadır.

| C# | VB.NET |
| --- | --- |
| **int** j = 12;  Type t1 = j.GetType();  ArrayList list = **new** ArrayList();  Type t2 = list.GetType();  Console.WriteLine("{0}: {1}: {2}",   t1.Name,   t1.FullName,   t1.AssemblyQualifiedName);  Console.WriteLine("{0}: {1}: {2}",   t2.Name,   t2.FullName,   t2.AssemblyQualifiedName); | **Dim** j **As** **Integer** = 12  **Dim** t1 **As** Type = j.**GetType**()  **Dim** list **As** ArrayList()  list = **New** ArrayList()  **Dim** t2 **As** Type = list.**GetType()**  Console.WriteLine("{0}: {1}: {2}",   t1.Name,   t1.FullName,   t1.AssemblyQualifiedName)  Console.WriteLine("{0}: {1}: {2}",   t2.Name,   t2.FullName,  t2.AssemblyQualifiedName) |

Kod 141, System.Object.GetType Kullanımı

Kod çıktısı aşağıda gösterilmiştir.

Int32: System.Int32: System.Int32, mscorlib, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089

ArrayList: System.Collections.ArrayList: System.Collections.ArrayList, mscorlib,

Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089

Tip Sınıfı ile Tip Bilgisine Erişim

Type.GetType metodu ile karakter dizi olarak ifade edilen tip adından Type nesnesi oluşturmak mümkündür.

İletilen tip adı, bütünleşik veya Assembly bütünleşik tip adı olabilir. Elde edilmek istenen tip aynı Assembly içerisinde ise Assembly bütünleşik adını kullanmaya gerek yoktur.

GetType metodu, parametre olarak iletilen tip adından Type nesnesi oluşturamadığı durumda null değeri (VB.NET için nothing) döndürmektedir ve tip adlarında büyük – küçük harf duyarlılığı bulunmaktadır. Bu davranış yüklü (ing. Overloaded) sürümlerini kullanarak değiştirilebilir.

C#

// System.Int32 karakter dizi değerinden çalışma zamanında tip nesnesi   
// oluştur.

Type t1 = Type.GetType("System.Int32");

// Geçersiz tip adı iletildiğinde istisna fırlat.

Type t2 = Type.GetType("System.Int32", **true**);

// Geçersiz tip adı iletildiğinde istisna fırlat ve tip adında büyük - küçük   
// harf duyarlılığı kullanma.

Type t3 = Type.GetType("System.Int32", **true**, **true**);

VB.NET

' System.Int32 karakter dizi değerinden çalışma zamanında tip nesnesi   
' oluştur.

**Dim** t1 **As** Type = Type.**GetType**("System.Int32")

' Geçersiz tip adı iletildiğinde istisna fırlat.

**Dim** t2 **As** Type = Type.**GetType**("System.Int32", **True**)

' Geçersiz tip adı iletildiğinde istisna fırlat ve tip adında

' büyük - küçük harf duyarlılığı kullanma.

**Dim** t3 **As** Type = Type.**GetType**("System.Int32", **True**, **True**)

Kod 142, System.Type.GetType Metodu

Rezerve Kelimeler ile Tip Bilgisine Erişim

Typeof ve GetType (sırasıyla C# ve VB.NET), tip tanımını kullanarak tip nesnesi elde etmeyi sağlayan rezerve kelimelerdir.

| C# | VB.NET |
| --- | --- |
| Type t1 = **typeof**(**int**);  Type t2 = **typeof**(ArrayList); | **Dim** t1 **As** Type = **GetType**(**Integer**)  **Dim** t2 **As** Type = **GetType**(ArrayList) |

Kod 143, Rezerve Kelimeler ile Tip Bilgisine Erişim

Assembly Sınıfı Üyeleri ile Tip Bilgisine Erişim

System.Reflection.Assembly sınıfı, dinamik olarak Assembly dosyalarının yüklenmesi ve tanımlı tip bilgilerine erişim için üyeler içermektedir.

Assembly.GetTypes, Assembly içerisindeki tip tanımlarını Type nesne dizisi olarak döndüren metottur.

| C# | VB.NET |
| --- | --- |
| Assembly a;  a = Assembly.GetExecutingAssembly();  Type[] types = a.GetTypes();  **foreach** (Type t **in** types) {  Console.WriteLine(t.FullName);  } | **Dim** a **As** Assembly  a = Assembly.GetExecutingAssembly()  **Dim** types() **As** Type = a.GetTypes()  **For** **Each** t **As** Type **In** types  Console.WriteLine(t.FullName)  **Next** |

Tablo 55, Assembly İçerisinde Tanımlı Tip Bilgilerine Erişim

Assembly.GetType metodu, Assembly içerisinde tanımlı bir tipe bütünleşik tip adı ile erişebilmeyi sağlamaktadır.

| C# | VB.NET |
| --- | --- |
| Assembly a;  a = Assembly.GetExecutingAssembly();  Type t = a.GetType("Ornekler.Test");  Console.WriteLine(t.FullName); | **Dim** a **As** Assembly  a = Assembly.GetExecutingAssembly()  **Dim** t **As** Type  t = a.GetType("Ornekler.Test")  Console.WriteLine(t.FullName) |

Kod 144, Assembly İçerisinde Tanımlı Tipe Erişim

Assembly.GetType metoduna bütünleşik tip adı parametre olarak iletilerek tip nesnesi elde edilmektedir. Aynı işlev, Type.GetType metoduna tipin Assembly bütünleşik adının iletilmesi ile de sağlanabilir.

Tip Sınıfı

Belirtilen yöntemler ile tip bilgisi Type nesnesi olarak elde edildikten sonra Type sınıfı üyeleri ile tip hakkında her tür bilgiye ulaşmak mümkündür.

Örneğin tipin sınıf tipi veya sayılabilir bir tip olduğu, tipe ait üyeler ve üyeler hakkında detaylı özellikler gibi tip tanımı ile ilgili her tür bilginin alınması mümkündür.

Aşağıda Type sınıfı temel üyeleri ve açıklamalar gösterilmiştir.

Aşağıda listelenen üye bilgileri ve Assembly sınıfı ile Assembly dosyalarını dinamik olarak yükleyerek, .NET Assembly'leri içerisindeki tip tanımlarını tüm üye bilgileri ile listeleyen tip görüntüleyici uygulamaları geliştirilebilir. Ücretsiz olarak sunulan Lutz Reflector bu araçlardan birisidir.

| Üye Adı | Açıklama |
| --- | --- |
| Assembly | Tipin tanımlı olduğu Assembly nesnesini belirtir özelliktir. |
| AssemblyQualifiedName | Tipin Assembly bütünleşik adını belirtir özelliktir. |
| BaseType | Tipin miras aldığı tipi belirtir özelliktir. |
| FullName | Tipin bütünleşik adını belirtir özelliktir.. |
| IsClass  IsInterface  IsSealed  IsPublic  IsEnum  IsGenericType  IsPrimitive  … | Tip niteliğini (Sınıf, arabirim, jenerik vb.) elde etmek için kullanılması gereken özelliklerdir. |
| IsVisible | Tipin Assembly dışında kullanılabilmesi durumunda True, sadece Assembly içerisinde kullanılabilmesi durumunda False değerini döndürür özelliktir. |
| GetConstructor | Tipe ait yapıcı metodu almak için kullanılması gereken metottur. |
| GetConstructors | Tipin yapıcı metotlarını almak için kullanılması gereken metottur. |
| GetEvents | Tip ile ilişkili olayları almak için kullanılması gereken metottur. |
| GetMethods | Tipe ait metotları almak için kullanılması gereken metottur. |
| GetProperties | Tipe ait özellikleri almak için kullanılması gereken metottur. |

Tablo 56, System.Type Sınıfı Üye Bilgileri

Dinamik Nesne Oluşturma

Çalışma zamanında elde edilen tip bilgisi ile, ilgili tipe ait nesneler oluşturmak mümkündür.

Derleme zamanında, Assembly referansı ekleyerek new rezerve kelimesi ile nesne oluşturma erken bağlama (ing. Early Binding) olarak tanımlanmaktadır.   
  
Derleme zamanında tip ile ilgili herhangi bir bilgi olmadan çalışma zamanında tip bilgisinin elde edilmesi ve böylece nesne oluşturma geç bağlama (ing. Late Binding) olarak tanımlanmaktadır.

System.Activator Sınıfı

System.Activator sınıfı, geç bağlama yöntemi ile çalışma zamanında nesne oluşturmak amacıyla kullanılan sınıftır.

Activator.CreateInstance, parametre olarak iletilen Assembly adı, tip nesnesi veya tip adı ile nesne oluşturmayı sağlayan statik metottur.

Aşağıda çeşitli kullanım örnekleri ve açıklamalar gösterilmiştir.

C#

// DotNetKitabı.Lib isimli Assembly'nin yüklenmesi.

Assembly a = Assembly.Load("DotNetKitabı.Lib");

// DotNetKitabı.Lib isimli Assembly içerisinde tanımlı Test sınıfı tip   
// bilgisine erişim.

Type t1 = a.GetType("Ornekler.Test");

**object** o1;

// Tip nesnesini parametre olarak ileterek nesne oluşturulmaktadır.

o1 = Activator.CreateInstance(t1);

// Yukarıdaki işlem aşağıda, CreateInstance metoduna Assembly ve tip adı   
// parametre olarak iletilerek yapılmaktadır.

**object** o2;

o2 = Activator.CreateInstance("DotNetKitabı.Lib", "Ornekler.Test");

// CreateInstanceFrom metodu ile Assembly yolunun belirtilmesi mümkündür.

**object** o3;

o3 = Activator.CreateInstanceFrom(@"c:\DotNetKitabı.Lib.dll", "Ornekler.Test");

// Aşağıdaki örnekte Test sınıfının tam sayı ve karakter dizi değerlerini  
// parametre olarak alan yapıcı metodu çağrılacaktır. Parametrelerin   
// belirtilmemesi durumunda nesneye ait ön tanımlı yapıcı metot   
// çağrılmaktadır.

**object**[] cParams;

cParams = **new** **object**[] {1, ""};

**object** o4;

o4 = Activator.CreateInstance(t1, cParams);

VB.NET

' DotNetKitabı.Lib isimli Assembly'nin yüklenmesi.

**Dim** a **As** Assembly = Assembly.Load("DotNetKitabı.Lib")

' DotNetKitabı.Lib isimli Assembly içerisinde tanımlı Test sınıfı tip   
' bilgisine erişim.

**Dim** t1 **As** Type = a.**GetType**("Ornekler.Test")

**Dim** o1 **As** **Object**

' Tip nesnesini parametre olarak ileterek nesne oluşturulmaktadır.

o1 = Activator.CreateInstance(t1)

' Yukarıdaki işlem aşağıda, CreateInstance metoduna Assembly ve tip adı   
' parametre olarak iletilerek yapılmaktadır.

**Dim** o2 **As** **Object**

o2 = Activator.CreateInstance("DotNetKitabı.Lib", "Ornekler.Test")

' CreateInstanceFrom metodu ile Assembly yolunun belirtilmesi mümkündür.

**Dim** o3 **As** **Object**

o3 = Activator.CreateInstanceFrom("c:\DotNetKitabı.Lib.dll", "Ornekler.Test")

' Aşağıdaki örnekte Test sınıfının tam sayı ve karakter dizi değerlerini  
' parametre olarak alan yapıcı metodu çağrılacaktır. Parametrelerin   
' belirtilmemesi durumunda nesneye ait ön tanımlı yapıcı metot   
' çağrılmaktadır.

**Dim** cParams() **As** **Object** = {1, ""}

**Dim** o4 **As** **Object**

o4 = Activator.CreateInstance(t1, cParams)

Öznitelikler

Öznitelikler (ing. Attributes), programcı tarafından herhangi bir Assembly, tip, metot veya alan gibi yapılara ek nitelikler kazandırabilmeyi sağlamaktadır.

Temel Bilgiler

Kitabın daha önceki bölümlerinde Assembly öznitelikleri hakkında bilgiler sunulmuştu. Dikkat edilirse bu öznitelikler ile sağlanmaya çalışılan işlev, Assembly'ye yeni nitelikler kazandırmaktır. Örneğin AssemblyCompany isimli öznitelik ile Assembly'nin üretici firması belirtilmektedir.

Özniteliklerin pratikte kullanımı ile ilgili bir diğer örnek Obsolete isimli özniteliktir. Bu öz niteliğin amacı, derleyiciye ilgili yapının kullanımının önerilmediğini iletmektedir. Herhangi bir yapının bu öznitelik ile donatılması durumunda derleyici otomatik olarak uyarı vermektedir.

| C# | VB.NET |
| --- | --- |
| **namespace** DotNetKitabı.Attributes {  [Obsolete]  **class** Test {  **public** **enum** Renk {  Siyah,  Beyaz  }  }  **class** Program {  **static** **void** Main() {  Test t = **new** Test();  }  }  } | **Namespace** DotNetKitabı.Attributes  <Obsolete()> \_  **Class** Test  **Public** **Enum** Renk  Siyah  Beyaz  **End** **Enum**  **End** **Class**  **Class** Program  **Shared** **Sub** Main()  **Dim** t **As** Test = **New** Test()  **End** **Sub**  **End** **Class**  **End** **Namespace** |

Kod 145, Obsolete Öz Niteliği ile Donatılma

Yukarıdaki örnekte Test isimli sınıf Obsolete öz niteliği ile donatılmış ve Main metodu içerisinde bu sınıf kullanılmıştır. Derlemenin yapılması ardından VS.NET uyarılar penceresi aşağıda gösterilmiştir.

Ekran Çıktısı 48, Obsolete Öz Niteliği ve Derleyici Uyarısı

Derleyici, bu öznitelik ile donatılmış herhangi bir yapının kullanımına rastladığında otomatik olarak uyarı üretmektedir. Yukarıdaki örnekte sağlanan işlev, Obsolete öz niteliği ile donatılan Test isimli sınıfa, programcı tarafından kullanımının önerilmediğini belirten yeni bir nitelik kazandırmaktır.

Obsolete öz niteliği .NET sınıf kütüphanesi içerisinde eski sürüm ve yeni sürüm arasında kullanımı önerilmeyen yapılar için sıklıkla kullanılmaktadır.

Öznitelik Donatımı

Öznitelik ile donatılma, tip tanımına ön ek olarak [] ve <> (sırasıyla C# ve VB.NET) karakterleri arasında öz niteliği belirtmektir. Bu durumda, ilgili tip öznitelik ile donatılmakta ve derleme ardından ilgili tip içerisine öznitelik donatımı eklenmektedir.

Herhangi bir yapının birden çok öznitelik ile donatılması mümkündür.

| C# | VB.NET |
| --- | --- |
| [Serializable]  **class** Test {  [Obsolete]  [NonSerialized]  **private** **string** s;  } | <Serializable()> \_  **Class** Test  <Obsolete(), NonSerialized()> \_  **Private** s **As** **String**  **End** **Class** |

Kod 146, Birden Çok Öznitelik ile Donatılma

Yukarıdaki örnekte s isimli alan, Obsolete ve NonSerialized özniteliklerinin her ikisi ile birlikte donatılmıştır.

Öznitelikler, parametre kullanımını desteklemektedir.

| C# | VB.NET |
| --- | --- |
| [Obsolete("Kullanmayınız", true)]  **class** Test {  } | <Obsolete("Kullanmayınız", **True**)> \_  **Class** Test  **End** **Class** |

Kod 147, Yapıcı Metot Giriş Niteleyici Örnekleri

Yukarıdaki örnekte Obsolete özniteliği iki parametre iletilerek kullanılmaktadır. İlk parametre derleyici tarafından gösterilecek mesajı, ikinci parametre ise uyarı yerine derleme hatası verilmesi gerektiğini belirtmektedir.

Ekran Çıktısı 49, Obsolete Öz Niteliği ve Derleyici Hatası

Öznitelik Sınıfı ve Nesneleri

.NET içerisinde kullanılan her yapıda olduğu gibi öznitelikler de nesnelerdir. Öznitelikler, System.Attribute sınıfını miras alan nesnelerdir.

C#

**public** **sealed** **class** ObsoleteAttribute : System.Attribute {

**public** ObsoleteAttribute();

**public** ObsoleteAttribute(**string** message);

**public** ObsoleteAttribute(**string** message, **bool** error);

}

VB.NET

**Public** **NotInheritable** **Class** ObsoleteAttribute

**Inherits** System.Attribute

**Public** **Sub** **New**()

**Public** **Sub** **New**(**ByVal** message **As** **String**)

**Public** **Sub** **New**(**ByVal** message **As** **String**, **ByVal** [**error**] **As** **Boolean**)

**End** **Class**

Kod 148, Obsolete Öznitelik Tanımı

C# ve VB.NET derleyicisi, kolaylık sağlamak amacıyla Attribute son ekinin kullanımını zorunlu kılmamaktadır. Aşağıdaki kullanım birebir diğer örneklerdeki kullanımlar ile aynı işleve sahiptir.

| C# | VB.NET |
| --- | --- |
| [ObsoleteAttrbiute]  **class** Test {  } | <ObsoleteAttribute()> \_  **Class** Test  **End** **Class** |

Herhangi bir yapının öznitelik ile donatılması durumunda derleyici tarafından arka planda yapılan işlem, öznitelik nesnesi oluşturmak ve oluşturulan öznitelik nesnesini yapı tanımına bir nitelik olarak eklemektir. Yukarıdaki Obsolete öz niteliği örneğinde, ObsoleteAttribute sınıfının parametresiz yapıcı metodu çağrılarak öznitelik nesnesi oluşturulmaktadır.

Genellemek gerekirse, normal nesne oluşturma işlemi new rezerve kelimesi ile yapılırken, öznitelik nesnesi oluşturma işlemi [] ve <> (sırasıyla C# ve VB.NET) karakterleri ile yapılmaktadır.

Öznitelik Özellikleri

Öznitelik nesneleri, normal nesnelerde olduğu gibi, sınıf yapıcı metotları çağrılarak oluşturulmaktadır. Farklı parametrelere sahip yapıcı metotlar çağrılarak öznitelik nesnesi oluşturulabilir. Daha önce sunulan örnekte Obsolete öz niteliği, hata mesajı ve uyarı yerine derleme hatası verilmesi gerektiğini belirtir şekilde oluşturulmuştur.

Öznitelikler System.Attribute sınıfını miras alan sınıflar olduğuna göre, öznitelikler için de sınıf ve üye tanım kuralları geçerlidir. Bu çerçevede, özniteliklere özellikler veya metotlar eklemek mümkündür.

Yapıların öznitelikler ile donatılması farklı parametreler içeren yapıcı metotların çağrımı ile yapılabileceği gibi, donatılma aşamasında öznitelik özellikleri de belirtilebilir.

C#

**public** **class** XmlRootAttribute: System.Attribute {

**public** XmlRootAttribute();

**public** XmlRootAttribute(**string** elementName);

**public** **string** DataType { **get**; **set**; }

**public** **string** ElementName { **get**; **set**; }

**public** **bool** IsNullable { **get**; **set**; }

**internal** **string** Key { **get**; }

}

VB.NET

**Public** **Class** XmlRootAttribute

**Inherits** System.Attribute

**Public** **Sub** **New**()

**Public** **Sub** **New**(**ByVal** elementName **As** **String**)

**Public** **Property** DataType **As** **String**

**Public** **Property** ElementName **As** **String**

**Public** **Property** IsNullable **As** **Boolean**

**Friend** **ReadOnly** **Property** Key **As** **String**

**End** **Class**

Kod 149, XmlRoot Öznitelik Tanımı

XML serilizasyon işleminde kök elemanı belirtmek için kullanılan System.Xml.Serialization.XmlRoot isimli öznitelik tanımı yukarıda gösterilmiştir. Öznitelik ile bir sınıfın donatılması aşamasında, öznitelik özellikleri aşağıdaki şekilde belirtilebilir.

C#

[XmlRoot(ElementName="xyz", IsNullable=**false**)]

**class** Test {

}

VB.NET

<XmlRoot(ElementName:="", IsNullable:=**False**)> \_

**Class** Test

**End** **Class**

Yukarıdaki örnek kullanımda, ElementName ve IsNullable özelliklerine değer ataması yapılmaktadır. Yapılan işlem, XmlRoot sınıfının parametresiz yapıcı metodunu çağırmak ve özelliklere değer atamaktır.

Öznitelik özelliklerine değer atama, parametre alan yapıcı metotlar ile birlikte de kullanılabilmektedir.

C#

[XmlRoot("elementName", IsNullable=**true**, DataType="string")]

**class** Test {

}

VB.NET

<XmlRootAttribute("elementName", IsNullable:=**True**, DataType:="**string**")> \_

**Class** Test

**End** **Class**

Örnekte, string tipinde parametre alan yapıcı metot kullanılmış ve IsNullable ile DataType özelliklerinin değerleri belirtilmiştir.

Öznitelik Hedefleri

Öznitelik hedefleri, öz niteliğin hangi yapılar için geçerli olduğunu veya hangi yapıların tanımlanan öznitelik ile donatılabileceğini belirtmektedir.

System.AttributeTargets sayılabilir tipi, öznitelik hedeflerini belirtmek için kullanılmaktadır.

| C# | VB.NET |
| --- | --- |
| **public** **enum** AttributeTargets {  All = 0x7fff,  Assembly = 1,  Class = 4,  Constructor = 0x20,  Delegate = 0x1000,  Enum = 0x10,  Event = 0x200,  Field = 0x100,  GenericParameter = 0x4000,  Interface = 0x400,  Method = 0x40,  Module = 2,  Parameter = 0x800,  Property = 0x80,  ReturnValue = 0x2000,  Struct = 8  } | **Public** **Enum** AttributeTargets  All = 32767  [Assembly] = 1  [**Class**] = 4  Constructor = 32  [**Delegate**] = 4096  [**Enum**] = 16  [**Event**] = 512  Field = 256  GenericParameter = 16384  [**Interface**] = 1024  Method = 64  [**Module**] = 2  Parameter = 2048  [**Property**] = 128  ReturnValue = 8192  Struct = 8  **End** **Enum** |

Kod 150, System.AttributeTargets Sayılabilir Tip Tanımı

Örneğin, XmlRoot öznitelik örneğinde, bu öz niteliğin metotlar için kullanımı anlam ifade etmemektedir. Bu durumda, öz niteliğin sadece donatılmasının anlamlı olacağı yapılarda kullanımı AttributeTargets sayılabilir tipi ile belirtilmektedir.

C#

[AttributeUsage(AttributeTargets.ReturnValue |   
 AttributeTargets.Interface | AttributeTargets.Enum |   
 AttributeTargets.Struct | AttributeTargets.Class)]

**public** **class** XmlRootAttribute : Attribute {

. . .

}

VB.NET

<AttributeUsage(AttributeTargets.ReturnValue \_

**Or** AttributeTargets.**Interface** \_

**Or** AttributeTargets.**Enum** \_

**Or** AttributeTargets.Struct \_

**Or** AttributeTargets.**Class**)> \_

**Public** **Class** XmlRootAttribute

**Inherits** Attribute

**. . .**

**End** **Class**

Yukarıda XmlRoot öznitelik sınıf tanımı gösterilmiştir. Hangi yapıların öznitelik ile donatılmasının anlamlı olacağı, öznitelik sınıf tanımının System.AttributeUsage öz niteliği ile donatılması ile belirtilmektedir.

AttributeUsage, öz niteliğin aynı yapı üzerinde birden fazla donatılabilme ve miraslık yönetiminin yapılabilmesi için sırasıyla AllowMultible ve Inheritable özelliklerini içermektedir.

Sayılabilir tiplerin ve, veya operatörleri ile kullanımı konusunda kitabın daha önceki bölümlerinde gerekli bilgiler bulunmaktadır.  
  
AttributeTargets.All değeri her tipin öznitelik ile donatılabileceğini ifade etmektedir.

Assembly'leri hedef olarak belirleyen özniteliklerin kullanımı, Assembly: kelimesinin öznitelik adına ön ek yapılması ile sağlanmaktadır. VS.NET projelerinde genel düzeni sağlamak amacıyla Assembly öznitelikleri AssemblyInfo.cs ve AssemblyInfo.vb (sırasıyla C# ve VB.NET) kaynak dosyalarında bulunmaktadır.

Örneğin

| C# | VB.NET |
| --- | --- |
| [Assembly**:** AssemblyVersion("1.0.0")] | <Assembly: AssemblyVersion("1.0.0")> |

Kod 151, Assembly Öznitelikleri Kullanımı

Programcı Tanımlı Öznitelik Sınıfları

Programcılar tarafından öznitelik tanımı, System.Attribute sınıfını miras alan yeni bir sınıf tanımlamak ve öznitelik hedeflerini belirtmek ile yapılabilir.

C#

[AttributeUsage(AttributeTargets.Method)]

**class** YetkiAttribute : Attribute {

**private** **string** yetki;

**public** YetkiAttribute(**string** yetki) {

**this**.yetki = yetki;

}

**public** **string** Yetki {

**get** {

**return** yetki;

}

}

}

VB.NET

<AttributeUsage(AttributeTargets.Method)> \_

**Class** YetkiAttribute

**Inherits** Attribute

**Private** \_yetki **As** **String**

**Public** **Sub** **New**(**ByVal** yetki **As** **String**)

\_yetki = yetki

**End** **Sub**

**Public** **ReadOnly** **Property** Yetki()**As** **String**

**Get**

**Return** \_yetki

**End** **Get**

**End** **Property**

**End** **Class**

Kod 152, Programcı Tanımlı Örnek Öznitelik

Yukarıdaki örnekte YetkiAttribute isminde, hedef olarak sadece metotları belirten öznitelik tanımı yapılmıştır. Tanımlanan öznitelik örnek kullanımı aşağıda gösterilmiştir.

| C# | VB.NET |
| --- | --- |
| **class** Test {  [Yetki("Dosya Silme")]  **public** **void** M() {  }  } | **Class** Test  <Yetki("Dosya Silme")> \_  **Public** **Sub** M()  **End** **Sub**  **End** **Class** |

Programcı tarafından tanımlanan öznitelik sınıflarının AttributeUsage öz niteliği ile donatılarak hangi hedefler için geçerli olduğunun belirtilmesi önerilmektedir.

.NET Sınıf Kütüphanesi Öznitelikleri

Bu konu başlığında, .NET sınıf kütüphanesi içerisinde tanımlı çeşitli öznitelikler ve özniteliklerin kullanım alanları hakkında bilgiler verilecektir.

Öznitelikler ve Açıklamaları

Aşağıda önemli görülen bazı öznitelikler ve açıklamaları gösterilmiştir.

| Öznitelik | Açıklama |
| --- | --- |
| DllImport *[[8]](#footnote-8)* | .NET uygulamaları içerisinden standart Win32 DLL kütüphanelerinin çağrımını sağlamak amacıyla kullanılmalıdır. |
| CLSCompliant | Donatıldığı yapının MSIL standartlarına tam olarak uyduğunu garanti altına almak için kullanılması gereken özniteliktir. |
| Serializable | Donatıldığı yapının serilize edilebilirliğini sağlayan özniteliktir. |
| WebMethod | Donatıldığı metodun XML web hizmet çağrılarında kullanılabileceğini belirten özniteliktir. |
| Obsolete | Donatıldığı yapının kullanılmaması gerektiğini belirten özniteliktir. |

Tablo 57, .NET Sınıf Kütüphanesi Öznitelikleri ve Açıklamaları

Win32 DLL Dosyalarının Kullanımı

Yönetilen kod içerisinden farklı programlama dilleri (C, C++, VB, Delphi, ...) ile derlenmiş yönetilmeyen Win32 DLL dosyalarının kullanımı DllImport öz niteliği ile sağlanabilmektedir.

Bu öznitelik aynı zamanda Windows API çağrımlarında da kullanılabilmektedir.

**Win32 DLL Hazırlama**

Aşağıda, örnek olması açısından, Borland Delphi ile hazırlanmış basit DLL kaynak kodu gösterilmiştir.

**library** OrnekDLL;

**uses**

SysUtils,

Classes;

{$R \*.res}

**function** Topla(x, y: Integer): Integer; **cdecl**; **safecall**;

**begin**

Result := x + y;

**end**;

**exports**

Topla;

**begin**

**end**.

Kaynak kodun Delphi ile derlenmesi ardından OrnekDLL.dll isminde Win32 DLL dosyası oluşacaktır. Topla isimli fonksiyonun dışarıdan çağrılabilir şekilde tanımlandığına dikkat edilmelidir.

DLL dosyası uygulama ile aynı klasöre, Windows veya System32 klasörüne kopyalanabilir.

**DLL Fonksiyonlarının Kullanımı**

Sistemde yüklü yönetilmeyen Win32 DLL dosyaları içerisinden fonksiyonların çağrılabilmesi için wrapper adı verilen bir sınıf tanımlanmalı, sınıfa ait statik metot DllImport öz niteliği ve extern rezerve kelimesi ile donatılmalıdır.

Aşağıda C# ve VB.NET için gerekli kod örnekleri gösterilmiştir.

C#

**class** OrnekDll {

[DllImport("OrnekDLL.dll")]

**public** **static** **extern** **int** Topla(**int** x, **int** y);

}

VB.NET

**Class** OrnekDll

**Public** **Declare** **Function** Topla **Lib** "OrnekDLL.dll" (**ByVal** x **As** **Integer**,   
 **ByVal** y **As** **Integer**)   
 **As** **Integer**

**End** **Class**

Yapılan tanımlamalar ile Win32 DLL fonksiyonu program içerisinden aşağıdaki gibi kullanılabilir.

| C# | VB.NET |
| --- | --- |
| **namespace** DotNetKitabı.Attributes {  . . .  **class** Program {  **static** **void** Main() {  **int** sonuc = OrnekDll.Topla(3, 4);  Console.WriteLine(sonuc);  }  }  } | **Namespace** DotNetKitabı.Attributes  . . .  **Class** Program  **Shared** **Sub** Main()  **Dim** sonuc **As** **Integer**  sonuc = OrnekDll.Topla(4, 4)  Console.WriteLine(sonuc)  **End** **Sub**  **End** **Class**  **End** **Namespace** |

Kod 153, Win32 DLL Kullanım Örneği

Öznitelik Kullanımı

Farklı yapıların özniteliklerle donatılması, öznitelik nesneleri ve programcı tarafından öznitelik sınıfları tanımlanması hakkında önceki konu başlıklarında bilgiler sunulmuştur.

Bilindiği gibi herhangi bir yapının öznitelik ile donatılması durumunda, öznitelik nesnesi oluşturulmakta ve oluşturulan öznitelik nesnesi tip tanımı içerisinde saklanmaktadır. Bir başka anlatımla, öznitelik nesnesi tipin bir parçası olmaktadır.

Programcı tarafından öznitelik sınıfları tanımlanması ve kullanımı ile sağlanabilecek işlevlerden birisi de tip tanımlarına ek bilgiler eklenmesini sağlamaktır.

Tip tanımı ile ifade edilmek istenen sınıf veya arabirim tanımları ile sınırlı değildir. Özellik, metot, delege veya olay tanımları da tip tanımları olarak algılanmalıdır.

Yansıma ve Öznitelikler

Çalışma zamanında yansıma yardımıyla herhangi bir tip için, tipin donatıldığı özniteliklerin alınması mümkündür.

Type.GetCustomAttributes metodu ile yapılan bu işlem ile ilgili örnek gösterilmiştir.

| C# | VB.NET |
| --- | --- |
| **static** **void** Listele(Type t) {  **object**[] list;  list = t.GetCustomAttributes(**false**);  **foreach** (**object** o **in** list) {  Console.WriteLine(o);  }  } | **Shared** **Sub** Listele(**ByVal** t **As** Type)  **Dim** list() **As** **Object**  list = t.GetCustomAttributes(**False**)  **For** **Each** o **As** **Object** **In** list  Console.WriteLine(o)  **Next**  **End** **Sub** |

Kod 154, Tipin Donatıldığı Özniteliklerin Alınması

Yukarıda tanımlanan Listele metodu, parametre olarak iletilen tip içerisinde bulunan öznitelik nesnelerini listelemektedir.

Type.GetCustomAttributes metoduna true değerinin parametre olarak iletimi, tipin miras aldığı tipler için tanımlı özniteliklerin de listelenmesini sağlamaktadır.

| C# | VB.NET |
| --- | --- |
| // Test sınıfı üç öznitelik ile // donatılmaktadır.  [XmlRoot(DataType="string")]  [Serializable]  [Obsolete("Kullanmayınız")]  **class** Test {  }  **class** Program {  . . .  **static** **void** Main(**string**[] args) {  // typeof rezerve kelimesi ile  // test isimli sınıfa ait tip   // nesnesi elde edilmekte ve  // parametre olarak iletilmektedir.  Listele(**typeof**(Test));  }  } | ' Test sınıfı üç öznitelik ile ' donatılmaktadır.  <XmlRoot(DataType:="**string**"), \_  Serializable(), \_  Obsolete("Kullanmayınız")> \_  **Class** Test  **End** **Class**  **Class** Program  . . .  **Shared** **Sub** Main(**ByVal** args()   **As** **String**)  ' GetType rezerve kelimesi ile  ' test isimli sınıfa ait tip   ' nesnesi elde edilmekte ve  ' parametre olarak iletilmektedir.  Listele(**GetType**(Test))  **End** **Sub**  **End** **Class** |

Tablo 58, Öznitelikler Listeleme Örneği

Kod çıktısı aşağıda gösterilmiştir.

System.Xml.Serialization.XmlRootAttribute

System.ObsoleteAttribute

System.SerializableAttribute

Yansıma bölümünde, tip nesnesi elde etme yöntemleri konusunda bilgiler verilmiştir. Çalışma zamanında Type.GetType veya rezerve kelimeler ile tip nesnesine erişim, tipin donatıldığı özniteliklerin listelenmesi için yeterlidir.  
  
Farklı Assembly dosyalarında tanımlı tiplerin hangi öznitelikler ile donatıldığının belirlenmesi için Assembly.GetTypes veya Assembly.GetType metotlarının kullanımı mümkündür.  
  
Özetlemek gerekirse, tip nesnesinin herhangi bir yöntem ile elde edilmesi, tipin donatıldığı özniteliklere erişim için yeterlidir.

Örnekte dikkat edilmesi gereken konulardan birisi, Type.GetCustomAttributes metodunun nesne listesi döndürdüğüdür. Döndürülen nesne listesi, donatılma sırasına göre, öznitelik nesnelerini ifade etmektedir. Liste üzerinde tip kontrolü ve dönüşümü yapılarak istenilen öznitelik nesnesine erişmek mümkündür.

C#

**public** **static** **void** Listele(Type t) {

**object**[] list;

list = t.GetCustomAttributes(**false**);

**foreach** (**object** o **in** list) {

// Tip kontrolü ile, öznitelik nesnesinin ObsoleteAttribute

// olduğu kontrol edilmektedir.

**if** (o **is** ObsoleteAttribute) {

ObsoleteAttribute attr;

// Tip dönüşümü yapılmaktadır.

attr = (ObsoleteAttribute)o;

Console.WriteLine(attr.Message);

}

}

}

VB.NET

**Public** **Shared** **Sub** Listele(**ByVal** t **As** Type)

**Dim** list() **As** **Object**

list = t.GetCustomAttributes(**False**)

**For** **Each** o **As** **Object** **In** list

' Tip kontrolü ile, öznitelik nesnesinin ObsoleteAttribute

' olduğu kontrol edilmektedir.

**If** **TypeOf** (o) **Is** ObsoleteAttribute **Then**

**Dim** attr **As** ObsoleteAttribute

' Tip dönüşümü yapılmaktadır.

attr = **CType**(o, ObsoleteAttribute)

Console.WriteLine(attr.Message)

**End** **If**

**Next**

**End** **Sub**

Kod 155, Tip İçerisinden Öznitelik Nesnesine Erişim

Listele metodunun yeniden düzenlenmiş sürümüne parametre olarak Test sınıfı iletildiğinde aşağıdaki çıktı üretilecektir.

Kullanmayınız

Elde edilen Obsolete özniteliğinin, Test tipinin donatıldığı ObsoleteAttribute öznitelik nesnesi olduğuna dikkat edilmelidir.

Assembly Öznitelikleri

Assembly seviyesinde belirtilen öznitelikler Assembly.GetCustomAttributes metodu ile elde edilebilir.

| C# | VB.NET |
| --- | --- |
| Assembly a;  a = Assembly.GetExecutingAssembly();  **object**[] list;  list = a.GetCustomAttributes(**false**);  **foreach** (**object** o **in** list) {  Console.WriteLine(o);  } | **Dim** a **As** Assembly  a = Assembly.GetExecutingAssembly()  **Dim** list() **As** **Object**  list = a.GetCustomAttributes(**False**)  **For** **Each** o **As** **Object** **In** list  Console.WriteLine(o)  **Next** |

Kod 156, Assembly Özniteliklerine Erişim

Assembly özniteliklerinin elde edilmesi ile, örneğin uygulamalarda hakkında bölümünde firma ve telif haklarının gösterimi mümkündür.

Örnek Senaryo

Bu bölümde, yansıma ve öznitelikler konularının örnek bir senaryo çerçevesinde somut kullanımları hakkında bilgiler verilecektir.

Örnek senaryo aşağıda sunulmuştur.

ABC firması yeni bir yazılım projesi yapmaya hazırlanmaktadır. Müşteri ile görüşmeler sonucu, kullanıcının yetkileri ölçüsünde bazı işlevlere erişebilmesinin gerektiği ortaya çıkmıştır. Müşteri, kullanıcıları tanımladıktan sonra kendisine sunulan yönetim modülü yardımıyla istediği kullanıcılara istediği hakları verebilmeyi istemektedir.

Proje yazılım mimarı, .NET tarafından sağlanan yansıma ve öznitelikleri kullanarak müşteri talebinin kolaylıkla karşılanabileceğini değerlendirmektedir. Projenin gereksinim belirleme çalışmaları devam ederken, yazılım mimarı, yazılımın yetkilendirme alt yapısını tasarlamaya başlamıştır.

Tasarım, kullanıcının yazılımda yapabileceği işlemleri operasyon olarak değerlendirme ve her operasyonun programcı tarafından geliştirilecek sınıf metoduna karşılık gelmesini standart olarak belirlemektedir. Yazılımda olabilecek tüm işlemlerin operasyonlar şeklinde karakter dizi değerleri olarak veritabanında tanımlanması, böylece yönetici kullanıcının yönetim modülü ile istediği kullanıcıya istediği operasyon hakkını verebilmesi sağlanacaktır.

Kullanıcı tarafından yazılıma bir komut verilmesi durumunda (örneğin bir butona tıklaması), yazılım mimarı tarafından geliştirilen sınıfın ilgili metodunun çağrılması, bu metodun yapacağı yetki kontrolü ardından programcı tarafından geliştirilen metodun çağrılması ön görülmektedir.

Tasarım Detayları

Tasarımda kullanılan sınıflar ve açıklamaları aşağıda listelenmiştir.

| Tip | Assembly | Amaç |
| --- | --- | --- |
| Müşteri | Model.dll | Müşteri bilgisini ifade eden sınıftır. |
| MüşteriListesi | Model.dll | Müşteri listesini ifade eden sınıftır. |
| Operasyon | Model.dll | Kullanıcının yapabileceği işlemleri ve işlemler ile ilgili bilgileri ifade eden sınıftır. |
| OperasyonListesi | Model.dll | Operasyon listesini ifade eden sınıftır. |
| YetkiAttribute | Model.dll | Programcıların geliştirdikleri metotları donatacakları öznitelik sınıfıdır. |
| MüşteriİşSınıfı | İş.dll | Müşteri ile ilgili işlemlerin kod gerçekleştirimlerinin toplandığı sınıftır. |
| İşlemYöneticisi | UI.dll | Kullanıcı işlemlerinin yönetimi ve yetki kontrolünün yapıldığı sınıftır. |
| AnaPencere | UI.dll | Kullanıcı ara yüz sınıfıdır. |

Tablo 59, Yansıma ve Öznitelikler Senaryo Tipleri

Kullanılan yetkilendirme alt yapısı için, kullanıcının yapabileceği dört operasyon örnek olarak kullanılacaktır.

1. Müşteri Kaydı Ekleme
2. Müşteri Kaydı Silme
3. Müşteri Adına Göre Arama
4. Tüm Müşterileri Listeleme

Hedeflenen yazılımın ana penceresi aşağıda gösterilmiştir.

Ekran Çıktısı 50, Yansıma ve Öznitelik Senaryo Ana Penceresi

Her operasyon, ileride detaylarıyla anlatılacağı üzere, aşağıdaki özelliklere sahiptir.

* Operasyonu tekil olarak belirten karakter dizi tipinde operasyon kodu.
* Operasyonun kullanıcıya gösterilmek amacıyla kullanılacak karakter dizi tipinde olan adı.
* Operasyon kod gerçekleştiriminin yapıldığı karakter dizi tipinde olan Assembly adı.
* Operasyon kod gerçekleştiriminin yapıldığı karakter dizi tipinde olan sınıf adı.
* Operasyon ile ilişkilendirilmiş karakter dizi tipinde metot adı.

Genel Mimari

Aşağıda genel işleyiş ve sınıf kullanım mimarisi şematik olarak gösterilmiştir.

Şekil 32, Yansıma ve Öznitelikler Mimari Şeması

Kullanıcı, form üzerinde herhangi bir butona tıkladığında, yani işlem yapmak istediğinde, İşlemYöneticisi.İşlemYap metodu operasyon kodu ve istenilen parametrelerle çağrılmaktadır.

İşlemYap metodu, operasyon kodu ile operasyon listesinde arama yapmakta ve operasyon ile ilgili ad, Assembly, sınıf ve metot bilgilerine ulaşmaktadır.

Dinamik Assembly kullanım yöntemleri yardımıyla operasyona ait Assembly çalışma zamanında yüklenmektedir.

Assembly içerisinden, operasyon tarafından belirtilen sınıf, metot ve metodun donatıldığı öznitelikler yansıma yardımıyla elde edilmektedir. Metodun donatıldığı YetkiAttribute ile belirtilen işleme kullanıcının yetkisi olduğu kontrol edilmektedir. Yetkisinin bulunmaması durumunda işlem durdurulmaktadır.

Yansıma ve dinamik nesne oluşturma ile, nesne oluşturulmakta, metot çağrılmakta ve döndürdüğü değer elde edilmektedir. Kullanıcıya gerekli bilgi mesajı gösterilerek işlem tamamlanmaktadır.

Sınıf Gerçekleştirim Detayları

Bu bölümde, yukarıda isimleri, amaçları ve ilişkileri ile genel mimarisi aktarılan sınıflar hakkında anahtar niteliğinde olan kod gerçekleştirim bilgileri sunulacaktır.

Model Gerçekleştirimi

Müşteri ve MüşteriListesi isimli sınıfların kod gerçekleştirimleri aşağıda gösterilmiştir.

| C# | VB.NET |
| --- | --- |
| **public** **class** Müşteri{  // Özellikler  **public** **long** TcNo ...  **public** **string** Ad ...  **public** **string** Soyad ...    // Metotlar  **public** Müşteri(**long** tcNo,   **string** ad,   **string** soyad) {  **this**.tcNo = tcNo;  **this**.ad = ad;  **this**.soyad = soyad;  }  } | **Public** **Class** Müşteri  ' Özellikler  **Public** **Property** TcNo() **As** **Long** ...  **Public** **Property** Ad() **As** **String** ...  **Public** **Property** Soyad() **As** **String** .  ' Metotlar  **Public** **Sub** **New**(**ByVal** tcNo **As** **Long**,  **ByVal** ad **As** **String**,   **ByVal** soyad   **As** **String**)  \_tcNo = tcNo  \_ad = ad  \_soyad = soyad  **End** **Sub**  **End** **Class** |
| **C#** | **VB.NET** |
| **using** System.Collections.ObjectModel;  **public** **class** MüşteriListesi : KeyedCollection<**long**, Müşteri> {  . . .  }  } | **Imports** System.Collections.ObjectModel  **Public** **Class** MüşteriListesi  **Inherits** KeyedCollection(  **Of** **Long**, Müşteri)  . . .  **End** **Class** |

KeyedCollection sınıfı kullanımı hakkında kitabın ilerleyen bölümlerinde gerekli bilgiler verilecektir. Bu sınıfın müşteri listesini ifade ettiğinin okuyucu tarafından bilinmesi yeterlidir.

Aşağıda Operasyon ve OperasyonListesi sınıfların kod gerçekleştirimleri bulunmaktadır.

Sınıfların miras alınamaz şekilde tanımlandığına dikkat edilmelidir. Böylece, kötü niyetli bir programcının sınıfları miras alabilmesi ve davranışlarını değiştirebilmesinin önüne geçilmektedir.

| C# | VB.NET |
| --- | --- |
| **public** **sealed** **class** Operasyon {  // Özellikler  **public** **string** Kod ...  **public** **string** Ad ...  **public** **string** Assembly ...  **public** **string** Sınıf ...  **public** **string** Metot ...    // Metotlar  Operasyon(**string** kod,   **string** ad,   **string** assembly,   **string** sınıf,   **string** metot) {  **this**.kod = kod;  **this**.ad = ad;  **this**.assembly = assembly;  **this**.sınıf = sınıf;  **this**.metot = metot;  }  } | **Public** **NotInheritable** **Class** Operasyon  ' Özellikler  **Public** **Property** Kod() **As** **String ...**  **Public** **Property** Ad() **As** **String ...**  **Public** **Property** Assembly() **As** **String**  **Public** **Property** Sınıf() **As** **String ..**  **Public** **Property** Metot() **As** **String ..**  ' Metotlar  **Friend** **Sub** **New**(**ByVal** kod **As** **String**,  **ByVal** ad **As** **String**,   **ByVal** assembly   **As String**,   **ByVal** sınıf   **As** **String**,   **ByVal** metot   **As** **String**)  \_kod = kod  \_ad = ad  \_assembly = assembly  \_sınıf = sınıf  \_metot = metot  **End** **Sub**  **End** **Class** |

OperasyonListesi sınıfı statik yapıcı metodu içerisinde örnek operasyonlar hazırlanmaktadır. Bu işlem, örneğin veritabanından operasyonların elde edilmesi ile yapılabilir.

C#

**public** **sealed** **class** OperasyonListesi: KeyedCollection<**string**, Operasyon> {

**private** **static** OperasyonListesi liste;

**static** OperasyonListesi() {

// Statik yapıcı metot içerisinde ön tanımlı operasyonlar   
 // oluşturulmaktadır.

liste = **new** OperasyonListesi();

liste.Add(**new** Operasyon("MüşteriEkle", "Müşteri Ekleme", "İş",   
 "İş.MüşteriİşSınıfı", "Ekle"));

liste.Add(**new** Operasyon("MüşteriAra", "Müşteri Arama", "İş",   
 "İş.MüşteriİşSınıfı", "Ara"));

liste.Add(**new** Operasyon("MüşteriSil", "Müşteri Silme", "İş",   
 "İş.MüşteriİşSınıfı", "Sil"));

liste.Add(**new** Operasyon("MüşteriListele", "Müşteri Arama", "İş",   
 "İş.MüşteriİşSınıfı", "Listele"));

}

// Statik olarak tanımlanan bu özellik ile, sınıf tarafından kontrol   
 // edilen listeye dışarıdan salt okunur erişim sağlanmaktadır.

**public** **static** OperasyonListesi Liste {

**get** {

**return** liste;

}

}

// Yapıcı metot gizli tanımlanarak, bu sınıfa ait nesne oluşturma  
 // engellenmiştir.

**private** OperasyonListesi() {

}

**protected** override **string** GetKeyForItem(Operasyon item) {

**return** item.Kod;

}

}

VB.NET

**Public** **NotInheritable** **Class** OperasyonListesi

**Inherits** KeyedCollection(**Of** **String**, Operasyon)

**Private** **Shared** \_liste **As** OperasyonListesi

' Statik yapıcı metot içerisinde ön tanımlı operasyonlar   
 ' oluşturulmaktadır.

**Shared** **Sub** **New**()

\_liste = **New** OperasyonListesi()

\_liste.Add(**New** Operasyon("MüşteriEkle", "Müşteri Ekleme", "İş",   
 "İş.MüşteriİşSınıfı", "Ekle"))

\_liste.Add(**New** Operasyon("MüşteriAra", "Müşteri Arama", "İş",   
 "İş.MüşteriİşSınıfı", "Ara"))

\_liste.Add(**New** Operasyon("MüşteriSil", "Müşteri Silme", "İş",   
 "İş.MüşteriİşSınıfı", "Sil"))

\_liste.Add(**New** Operasyon("MüşteriListele", "Müşteri Arama", "İş",   
 "İş.MüşteriİşSınıfı", "Listele"))

**End** **Sub**

' Statik olarak tanımlanan bu özellik ile, sınıf tarafından kontrol   
 ' edilen listeye dışarıdan salt okunur erişim sağlanmaktadır.

**Public** **Shared** **ReadOnly** **Property** Liste() **As** OperasyonListesi

**Get**

**Return** \_liste

**End** **Get**

**End** **Property**

' Yapıcı metot gizli tanımlanarak, bu sınıfa ait nesne oluşturma  
 ' engellenmiştir.

**Private** **Sub** **New**()

**End** **Sub**

**Protected** **Overrides** **Function** GetKeyForItem(**ByVal** item **As** Operasyon)   
 **As** **String**

**Return** item.Kod

**End** **Function**

**End** **Class**

Son olarak model içerisinde bulunan YetkiAttrbiute öznitelik sınıfı kod gerçekleştirimi bulunmaktadır.

C#

[AttributeUsage(AttributeTargets.Method)]

**public** **sealed** **class** YetkiAttribute: Attribute {

**private** **string** operasyon;

**public** YetkiAttribute(**string** yetki) {

**this**.operasyon = yetki;

}

**public** **string** Operasyon {

**get** {

**return** operasyon;

}

}

}

VB.NET

<AttributeUsage(AttributeTargets.Method)> \_

**Public** **NotInheritable** **Class** YetkiAttribute

**Inherits** Attribute

**Private** \_operasyon **As** **String**

**Public** **Sub** **New**(**ByVal** yetki **As** **String**)

\_operasyon = yetki

**End** **Sub**

**Public** **ReadOnly** **Property** Operasyon() **As** **String**

**Get**

**Return** \_operasyon

**End** **Get**

**End** **Property**

**End** **Class**

İş Katman Gerçekleştirimi

İş katmanında sadece MüşteriİşSınıfı isimli, müşteri işlemlerini içeren sınıf bulunmaktadır.

Belirlenen mimari model çerçevesinde, bu sınıf içerisinde, kullanıcının yapabileceği her bir işleme karşılık gelen yalnız bir metot bulunmaktadır. Operasyon metodu olarak ifade edilebilecek bu metot, Yetki özniteliği ile donatılmıştır. Böylece, ilgili metodun hangi operasyon yetkisine sahip kullanıcı tarafından çağrılabileceği belirlenmektedir.

C#

**using** Model**;**

**public** **class** MüşteriİşSınıfı {

**private** **static** MüşteriListesi liste = **new** MüşteriListesi();

[Yetki("Müşteri Ekleme")]

**public** **void** Ekle(Müşteri müşteri) {

**try** {

liste.Add(müşteri);

}

**catch** (ArgumentException) {

**throw** **new** ApplicationException(  
 "Aynı T.C. No ile müşteri bulunmaktadır");

}

}

[Yetki("Müşteri Arama")]

**public** MüşteriListesi Ara(**string** ad) {

**return** liste.Filtrele(ad);

}

[Yetki("Müşteri Silme")]

**public** **void** Sil(**long** tcNo) {

liste.Remove(tcNo);

}

[Yetki("Müşteri Arama")]

**public** MüşteriListesi Listele() {

**return** liste.Filtrele(**string**.Empty);

}

}

VB.NET

**Imports** Model

**Public** **Class** MüşteriİşSınıfı

**Private** **Shared** liste **As** MüşteriListesi = **New** MüşteriListesi()

<Yetki("Müşteri Ekleme")> \_

**Public** **Sub** Ekle(**ByVal** müşteri **As** Müşteri)

**Try**

liste.Add(müşteri)

**Catch** exc **As** ArgumentException

**Throw** **New** ApplicationException(  
 "Aynı T.C. No ile müşteri bulunmaktadır")

**End** **Try**

**End** **Sub**

<Yetki("Müşteri Arama")> \_

**Public** **Function** Ara(**ByVal** ad **As** **String**) **As** MüşteriListesi

**Return** liste.Filtrele(ad)

**End** **Function**

<Yetki("Müşteri Silme")> \_

**Public** **Sub** Sil(**ByVal** tcNo **As** **Long**)

liste.Remove(tcNo)

**End** **Sub**

<Yetki("Müşteri Arama")> \_

**Public** **Function** Listele() **As** MüşteriListesi

**Return** liste.Filtrele(**String**.Empty)

**End** **Function**

**End** **Class**

Uygulama yazılımında tüm müşterilerin listelenmesi ile herhangi bir müşterinin aranması aynı operasyon adı ile belirtilmektedir.

Kullanıcı Ara Yüz Katmanı ve İş Yönetici Sınıfı

Uygulama yazılımı ana penceresinde kullanıcının herhangi bir butona tıklaması durumunda, iş yönetici sınıfına ait İşlemYap metodu çağrılmaktadır.

Aşağıda, örnek olması açısından, kullanıcının Müşteri Ekle başlıklı butona tıklaması durumunda çalıştırılacak kod gösterilmiştir.

C#

**private** **void** btnMüşteriEkle\_Click(**object** sender, EventArgs e) {

**try** {

// Kullanıcı ara yüz kontrollerinden kullanıcının girdiği bilgiyi  
 // alarak yeni bir Müşteri nesnesi oluşturulmaktadır.

Müşteri m = **new** Müşteri(**long**.Parse(ctlMüşteriTcNo.Text),  
 ctlMüşteriAd.Text, ctlMüşteriSoyad.Text);

// İşlemYap metoduna, operasyon kodu ve istenilen parametreler   
 // iletilebilmektedir.

İşlemYöneticisi.İşlemYap("MüşteriEkle", m);

// Kullanıcıya işlemin başarıyla yapıldığına dair mesaj gösterilmektedir.

MessageBox.Show("Müşteri eklendi");

}

**catch** (Exception exc) {

// İşlem sırasında istisna fırlatılması durumunda, istisna yakalanmakta  
 // ve kullanıcıya gösterilmektedir.

MessageBox.Show(exc.Message);

}

}

VB.NET

**Private** **Sub** btnMüşteriEkle\_Click(**ByVal** sender **As** System.**Object**,   
 **ByVal** e **As** System.EventArgs)   
 **Handles** btnMüşteriEkle.Click

**Try**

' Kullanıcı ara yüz kontrollerinden kullanıcının girdiği bilgiyi  
 ' alarak yeni bir Müşteri nesnesi oluşturulmaktadır.

**Dim** m **As** Müşteri = **New** Müşteri(**Long**.Parse(ctlMüşteriTcNo.Text),   
 ctlMüşteriAd.Text, ctlMüşteriSoyad.Text))

' İşlemYap metoduna, operasyon kodu ve istenilen parametreler   
 ' iletilebilmektedir.

İşlemYöneticisi.İşlemYap("MüşteriEkle", m)

' Kullanıcıya işlemin başarıyla yapıldığına dair mesaj gösterilmektedir.

MessageBox.Show("Müşteri eklendi")

**Catch** exc **As** Exception

' İşlem sırasında istisna fırlatılması durumunda, istisna yakalanmakta  
 ' ve kullanıcıya gösterilmektedir.

MessageBox.Show(exc.Message)

**End** **Try**

**End** **Sub**

Mimarinin en kritik sınıfı olan İşlemYöneticisi ile aşağıda listelenen işlevler sağlanmaktadır.

1. Yansıma yardımıyla iş sınıfı ve metot tip bilgilerine erişim.
2. Yetki öznitelik nesnesine erişim.
3. Kullanıcının, yapmak istediği işlemeye yetkisinin olup olmadığının kontrolü.
4. İş nesnesinin oluşturulması ve metodun çağrılması

**Yetki Kontrolü**

İşlemYöneticisi isimli sınıf içerisinde yetki kontrolü amacıyla iki adet metot bulunmaktadır.

C#

// Kullanıcının parametre olarak iletilen operasyona yetkisinin olup   
// olmadığını döndürmektedir. Burada, sisteme giriş yapmış   
// kullanıcının yetkileri, örneğin, veritabanınadan elde edilebilir.

**private** **static** **bool** kullanıcıYetkisiVarMı(**string** operasyonAd) {

**return** **true**;

}

// Metot bilgisini parametre olan alan bu metot, metodun donatıldığı Yetki  
// öznitelik nesnesini kontrol etmektedir. Programcının geliştirdiği   
// metodu öznitelik ile donatmayı unutması durumunda gerekli istisna  
// fırlatılmaktadır.

**private** **static** **void** YetkiKontrol(MemberInfo m) {

**object**[] attrList = m.GetCustomAttributes(**false**);

YetkiAttribute yetki = **null**;

**foreach** (**object** o **in** attrList) {

**if** (o **is** YetkiAttribute) {

yetki = (YetkiAttribute)o;

**if** (!kullanıcıYetkisiVarMı(yetki.Operasyon))

**throw** **new** ApplicationException(  
 **string**.Format("{0} yetkiniz bulunmamaktadır", yetki.Operasyon));

}

}

**if** (yetki == **null**)

**throw** **new** ApplicationException("Metot yetki tanımı yapılmamıştır");

}

VB.NET

' Kullanıcının parametre olarak iletilen operasyona yetkisinin olup   
' olmadığını döndürmektedir. Burada, sisteme giriş yapmış   
' kullanıcının yetkileri, örneğin, veritabanınadan elde edilebilir.

**Private** **Shared** **Function** YetkisiVarMı(**ByVal** operasyonAd **As** **String**)   
 **As** **Boolean**

**Return** **True**

**End** **Function**

' Metot bilgisini parametre olan alan bu metot, metodun donatıldığı Yetki  
' öznitelik nesnesini kontrol etmektedir. Programcının geliştirdiği   
' metodu öznitelik ile donatmayı unutması durumunda gerekli istisna  
' fırlatılmaktadır.

**Private** **Shared** **Sub** YetkiKontrol(**ByVal** m **As** MemberInfo)  
 **Dim** attrList() **As** **Object**

attrList = m.GetCustomAttributes(**False**)

**Dim** yetki **As** YetkiAttribute = **Nothing**

**For** Each o **As** **Object** **In** attrList

**If** (**TypeOf** (o) **Is** YetkiAttribute) **Then**

yetki = **CType**(o, YetkiAttribute)

**If** **Not** YetkisiVarMı(yetki.Operasyon) **Then**

**Throw** **New** ApplicationException(  
 **String**.Format("{0} yetkiniz bulunmamaktadır", yetki.Operasyon))

**End** **If**

**End** **If**

**Next**

**If** (yetki **Is** **Nothing**) **Then**

**Throw** **New** ApplicationException("Metot yetki tanımı yapılmamıştır")

**End** **If**

**End** **Sub**

**Yansıma ve Metot Çağrısı**

İşlemYap metodu kod gerçekleştirimi aşağıda gösterilmiştir.

C#

**sealed** **class** İşlemYöneticisi {

. . .

// Metot işlem kodu ve istenilen sayıda parametre alabilmektedir. Metoda   
 // iletilen parametreler olduğu gibi iş sınıfının ilgili metoduna   
 // iletilmektedir.

**public** **static** **object** İşlemYap(**string** operasyonKodu,   
 **params** **object**[] parametreler) {

// Operasyon koduna göre operasyon nesnesine erişim.

Operasyon op = OperasyonListesi.Liste[operasyonKodu];

// Operasyon tarafından belirtilen Assembly dosyasını dinamik olarak  
 // yükleme.

Assembly a = Assembly.Load(op.Assembly);

// Operasyon tarafından belirtilen iş sınıfına erişim. True parametresi   
 // ile sınıf tanımına erişilememesi durumunda istisna fırlatılması   
 // sağlanmaktadır.

Type işSınıfı = a.GetType(op.Sınıf, **true**);

// Operasyon tarafından belirtilen metoda erişim.

MethodInfo işMetodu = işSınıfı.GetMethod(op.Metot);

// Aktif kullanıcının, metodun donatıldığı Yetki özniteliği ile  
 // belirtilen operasyona yetkisi olup olmadığının kontrolü.  
 // Kullanıcının yetkisinin bulunmaması durumunda istisna fırlatılmakta,  
 // böylece işlem durdurulmaktadır.

YetkiKontrol(işMetodu);

// Dinamik olarak iş sınıf nesnesinin oluşturulması.

**object** işNesnesi = Activator.CreateInstance(işSınıfı);

**try** {

// MethodInfo.Invoke metodu ile, ilk parametre ile belirtilen  
 // nesnenin metodu çağrılmaktadır. İşlemYap metoduna iletilen  
 // parametreler bu metoda iletilmektedir.

**return** işMetodu.Invoke(işNesnesi, parametreler);

}

**catch** (TargetInvocationException exc) {

// Yansıma ile bir üye çağrısında oluşan istisna   
 // TargetInvocationException ile yakalanmaktadır. InnerException,  
 // üyenin gerçekte fırlattığı istisnayı belirtmektedir.

**throw** exc.InnerException;

}

}

}

VB.NET

**NotInheritable** **Class** İşlemYöneticisi

. . .

' Metot işlem kodu ve istenilen sayıda parametre alabilmektedir. Metoda   
 ' iletilen parametreler olduğu gibi iş sınıfının ilgili metoduna   
 ' iletilmektedir.

**Public** **Shared** **Function** İşlemYap(**ByVal** operasyonKodu **As** **String**,   
 **ByVal** **ParamArray** parametreler() **As** **Object**)   
 **As** **Object**

' Operasyon koduna göre operasyon nesnesine erişim.

**Dim** op **As** Operasyon = OperasyonListesi.Liste(operasyonKodu)

' Operasyon tarafından belirtilen Assembly dosyasını dinamik olarak  
 ' yükleme.

**Dim** a **As** Assembly = Assembly.Load(op.Assembly)

' Operasyon tarafından belirtilen iş sınıfına erişim. True parametresi   
 ' ile sınıf tanımına erişilememesi durumunda istisna fırlatılması   
 ' sağlanmaktadır.

**Dim** işSınıfı **As** Type = a.**GetType**(op.Sınıf, **True**)

' Operasyon tarafından belirtilen metoda erişim.

**Dim** işMetodu **As** MethodInfo = işSınıfı.GetMethod(op.Metot)

' Aktif kullanıcının, metodun donatıldığı Yetki özniteliği ile  
 ' belirtilen operasyona yetkisi olup olmadığının kontrolü.  
 ' Kullanıcının yetkisinin bulunmaması durumunda istisna fırlatılmakta,  
 ' böylece işlem durdurulmaktadır.

YetkiKontrol(işMetodu)

' Dinamik olarak iş sınıf nesnesinin oluşturulması.

**Dim** işNesnesi **As** **Object =** Activator.CreateInstance(işSınıfı)

**Try**

' MethodInfo.Invoke metodu ile, ilk parametre ile belirtilen  
 ' nesnenin metodu çağrılmaktadır. İşlemYap metoduna iletilen  
 ' parametreler bu metoda iletilmektedir.

**Return** işMetodu.Invoke(işNesnesi, parametreler)

**Catch** exc **As** TargetInvocationException

' MethodInfo.Invoke metodu ile, ilk parametre ile belirtilen  
 ' nesnenin metodu çağrılmaktadır. İşlemYap metoduna iletilen  
 ' parametreler bu metoda iletilmektedir.

**Throw** exc.InnerException

**End** **Try**

**End** **Function**

**End** **Class**

Sonuç

Yansıma, geleneksel programlama yöntemlerinin çok ötesinde olan tasarım şablonlarının kolaylıkla uygulanabilmesini sağlamaktadır. Örnek senaryoda, kullanıcının butona tıklaması ile başlatılan işlemler tamamen yansıma ile yapılmaktadır.

Senaryoda sunulan mimari ile elde edilen kazanımlardan birisi de, programcılar arasında daha kolay iş bölümünün yapılabilmesi, programcıların farklı sınıf kütüphaneleri üzerinde farklı programlama dilleri ile çalışabilmelerinin sağlanmış olmasıdır. Örneğin satış işlemlerini içeren sınıflar farklı bir Assembly içerisinde toplanabilir ve yansıma yardımıyla rahatlıkla kullanılabilir.

Yansımanın sağladığı kazanımların yanında performans açısından negatif etkileri de olduğu unutulmamalıdır. Her yöntemde olduğu gibi, yansıma da, gerektiğinde kullanılmalıdır.

Listelenen kaynak kodlar \Bölüm 11\Senaryo klasöründe bulunmaktadır.

Öğrenme ilkesi insanın temiz karakterini ortaya çıkarmak, insanlara yeni yaşam vermek ve nihai iyiye ve doğruya ulaşmak demektir.

Konfüçyüs

Bölüm

12

Koleksiyonlar

K

oleksiyonlar, farklı nesneleri saklayabilen ve saklanan nesnelere farklı yöntemlerle erişebilmeyi sağlanan yapılardır. Kitabın daha önceki konularında aktarılan dizilere bazı yönleri ile benzerlik göstermelerine rağmen, dizi kullanımı ile hedeflenen aynı tipte olan nesneleri hafızada ardışık olarak saklayabilmek ve indeks yardımı ile her bir dizi içeriğine erişebilmektir.

Koleksiyonlar, benzer işlevi sağlamakla birlikte, çalışma zamanında koleksiyona nesne ekleme, sıralama, arama yapma gibi işlevleri de desteklemektedirler.

.NET Koleksiyon Mimarisi

Koleksiyonlar, programcıların sık ihtiyaç duyabilecekleri "dinamik olarak listeye nesne ekleme, eklenen nesnelere erişme, nesneleri listeleme ve var olan nesneleri silme" gibi işlevlere çözüm sunan yapılardır.

Koleksiyonların bir diğer kullanım amacı, kitabın ilerleyen konularında detayları ile aktarılacağı üzere, kullanıcı ara yüz kontrollerine veri bağlama işlevini standart bir yapıya oturtmaktır.

Arabirim Tanımları

Standartlaştırılmış mimarinin kurulabilmesi amacıyla System.Collections ad alanı içerisinde IEnumarable, ICollection, IDictionary ve IList arabirimleri tanımlanmıştır.

Şekil 33, Koleksiyon Arabirimleri

Yukarıda gösterilen arabirimler ile sağlanan, gerek .NET sınıf kütüphanesi içerisinde, gerek programcılar tarafından yeni koleksiyon sınıflarının tanımlanabilmesi, tanımlanan koleksiyon sınıflarına standart yöntemler ile erişimin sağlanabilmesidir.

Sayılabilme

IEnumerable arabirimi, sayılabilmeyi ifade eden temel koleksiyon arabirimidir.

| C# | VB.NET |
| --- | --- |
| **public** **interface** IEnumerable {  IEnumerator GetEnumerator();  } | **Public** **Interface** IEnumerable  **Function** GetEnumerator()   **As** IEnumerator  **End** **Interface** |

Kod 157, IEnumerable Arabirim Tanımı

Bu arabirim gerçekleştirimini yapan bir sınıf, içerisinde bulunan elemanların (nesnelerin) IEnumerator arabirimi yardımıyla sayılabileceğini belirtmektedir.

| C# | VB.NET |
| --- | --- |
| **public** **interface** IEnumerator {  **bool** MoveNext();  **object** Current { **get**; }  **void** Reset();  } | **Public** **Interface** IEnumerator  **Function** MoveNext() **As** **Boolean**  **Sub** Reset()  **ReadOnly** **Property** Current **As** **Object**  **End** **Interface** |

Kod 158, IEnumarator Arabirim Tanımı

IEnumerator.MoveNext metodu, koleksiyon içerisinde bulunan bir sonraki nesnenin Current metodu ile döndürülmesi gerektiğini belirtmektedir. Metodun false döndürmesi, koleksiyon içerisinde sayılabilecek nesne kalmadığını, true döndürmesi sayılabilecek daha fazla nesne olduğunu ifade etmektedir.

IEnumerator.Reset metodu, koleksiyon sayım işleminin baştan itibaren yapılabilmesini sağlamaktadır.

C#

**class** TestEnumerator : IEnumerator {

**private** **int** index = 0;

**private** **string** str;

**public** **object** Current { **get** { **return** str[index]; } }

**public** **bool** MoveNext() {

**if** (index < str.Length - 1) {

index++;

**return** **true**;

}

**return** **false**;

}

**public** **void** Reset() {

index = -1;

}

**public** TestEnumerator(**string** str) {

**this**.str = str;

Reset();

}

}

**class** TestEnumerable : IEnumerable {

**public** IEnumerator GetEnumerator() {

**return** **new** TestEnumerator("DotNet");

}

}

VB.NET

**Class** TestEnumerator

**Implements** IEnumerator

**Private** index **As** **Integer** = 0

**Private** str **As** **String**

**Public** **ReadOnly** **Property** Current() **As** **Object** **Implements** IEnumerator.Current

**Get**

**Return** str(index)

**End** **Get**

**End** **Property**

**Public** **Function** MoveNext() **As** **Boolean** **Implements** IEnumerator.MoveNext

**If** index < str.Length - 1 **Then**

index = index + 1

**Return** **True**

**End** **If**

**Return** **False**

**End** **Function**

**Public** **Sub** Reset() **Implements** IEnumerator.Reset

index = -1

**End** **Sub**

**Public** **Sub** **New**(**ByVal** str **As** **String**)

**Me**.str = str

Reset()

**End** **Sub**

**End** **Class**

**Class** TestEnumerable

**Implements** IEnumerable

**Public** **Function** GetEnumerator() **As** IEnumerator  
 **Implements** IEnumerable.GetEnumerator

**Return** **New** TestEnumerator("DotNet")

**End** **Function**

**End** **Class**

Kod 159, Örnek IEnumerator ve IEnumerable Kullanımı

Yukarıdaki örnekte IEmumerator ve IEnumerable arabirim gerçekleştirimini yapan sırasıyla TestEnumerator ve TestEnumerable isimli sınıflar tanımlanmıştır.

TestEnumerator isimli sınıf, yapıcı metoduna iletilen karakter dizi değer üzerinde sayma işlemini yapmaktadır. Her iki sınıfın örnek kullanımı aşağıda gösterilmiştir.

C#

TestEnumerable test = **new** TestEnumerable();

IEnumerator e = test.GetEnumerator();

**while** (e.MoveNext()) Console.WriteLine(e.Current);

VB.NET

**Dim** test **As** TestEnumerable = **New** TestEnumerable()

**Dim** e **As** IEnumerator = test.GetEnumerator()

**While** e.MoveNext()

Console.WriteLine(e.Current)

**End** **While**

Örnek program çıktısı.

D  
o  
t  
N  
e  
t

Kitabın daha önceki konularında aktarılan foreach döngüsü IEnumerable arabirim gerçekleştirimini yapan sınıflar için kullanılabilmektedir. Yukarıda while döngüsü ile yapılan işlemin foreach döngüsüne dönüştürülmüş şekli aşağıdadır.

C#

TestEnumerable test = **new** TestEnumerable();

**foreach** (**object** o **in** test) { Console.WriteLine(o); }

VB.NET

**For** **Each** **Object** o **In** test

Console.WriteLine(o)

**Next**

C#, özelleştirilmiş sayılabilmeyi desteklemektedir. Örneğin  
  
 class Test  
 {  
 public IEnumerable GetList()  
 {  
 for (int i = 0; i < 5; i++)  
 yield return i;  
 }  
 }  
  
Yukarıdaki örnekte IEnumerable arabirimi döndüren GetList metodu içeriği istenildiği gibi düzenlenebilmektedir.   
  
 Test test = new Test();  
 foreach (object o in test.GetList())  
 {  
 Console.WriteLine(o);   
 // 0, 1, 2, 3, 4 çıktısı üretilecektir.  
 }  
  
Not: yield break ile koleksiyon sayma işlemi sonlandırılabilmektedir.

Temel Koleksiyon Arabirimi

IEnumerable arabirimini miras alan ICollection arabirimi temel koleksiyon arabirimidir.

C#

**public** **interface** ICollection: IEnumerable {

**void** CopyTo(Array array, **int** index);

**int** Count { **get**; }

**bool** IsSynchronized { **get**; }

**object** SyncRoot { **get**; }

}

VB.NET

**Public** **Interface** ICollection

**Implements** IEnumerable

**Sub** CopyTo(**ByVal** array **As** Array, **ByVal** index **As** **Integer**)

**ReadOnly** **Property** Count **As** **Integer**

**ReadOnly** **Property** IsSynchronized **As** **Boolean**

**ReadOnly** **Property** SyncRoot **As** **Object**

**End** **Interface**

Kod 160, ICollection Arabirim Tanımı

ICollection arabirimi üye bilgileri aşağıda gösterilmiştir.

| Üye Adı | Açıklama |
| --- | --- |
| Count | Koleksiyon boyutunu döndürmektedir. |
| IsSynchronized | Koleksiyonun senkronize olup olmadığını döndürür. [[9]](#footnote-9) |
| SyncRoot | Senkronize nesnesini döndürür. |
| CopyTo | Koleksiyon içeriğini dizi nesnesine kopyalar. |

Tablo 60, ICollection Üye Bilgileri

ICollection arabirimi ile hedeflenen amaç, farklı işlevlere sahip koleksiyon sınıflarını asgaride bu arabirim çatısı altında toplayabilmeyi sağlayabilmektedir.

Liste ve Sözlük Arabirimleri

IList ve IDictionary sınıfları, ICollection arabirimini miras alan, sırasıyla liste ve sözlük koleksiyon nesneleri için temel arabirimlerdir.

IList arabirimi, koleksiyona nesne ekleme, koleksiyon içerisinde arama yapma gibi çeşitli özellik ve metotları içermektedir.

| Üye Adı | Açıklama |
| --- | --- |
| IsFixedSize | Koleksiyonun tekrar boyutlandırılabilirliğini belirtmektedir. Değerinin true olması koleksiyon boyutunun sabit olduğunu belirtmektedir. |
| IsReadOnly | Koleksiyonun salt okunur olup olmadığını döndürmektedir. |
| Item | Koleksiyon içerisindeki herhangi bir nesneye indeks yardımıyla erişebilmeyi sağlayan metottur. |
| Add | Koleksiyona yeni bir nesne eklemek için kullanılması gereken metottur. |
| Clear | Koleksiyon içeriğini temizlemek (içeriğin yok edilmesi) amacıyla kullanılması gereken metottur. |
| Contains | Parametre olarak iletilen nesnenin koleksiyon içerisinde bulunup bulunmadığını döndüren metottur. |
| IndexOf | Parametre olarak iletilen nesnenin koleksiyon içerisindeki pozisyonunu döndürür. Nesnenin bulunamaması durumunda -1 değeri döndürülmelidir. |
| Insert | İstenilen pozisyona yeni bir nesne eklemek için kullanılmalıdır. |
| Remove | Parametre olarak iletilen nesneyi çıkarmak için kullanılmalıdır. |
| RemoveAt | İstenilen pozisyondaki nesneyi çıkarmak için kullanılmalıdır. |

Tablo 61, IList Üye Bilgileri

.NET sınıf kütüphanesinin bir parçası olarak System.Collections ad alanı içerisinde tanımlı ArrayList sınıfı IList gerçekleştirimini yapmaktadır.

IList arabirimi üyelerinin örnek kullanımı aşağıda gösterilmiştir.

| C# | VB.NET |
| --- | --- |
| // Yeni bir ArrayList nesnesi // oluştur.  ArrayList list = **new** ArrayList();  // Listeye yeni nesneler ekle.  list.Add("Audi");  list.Add("Mercedes");  // Liste boyutunu yazdır.  Console.WriteLine(list.Count); // 2  // Liste içerisinden nesneyi sil  list.Remove("Audi");  // 0 pozisyonuna yeni nesne ekle. list.Insert(0, "BMW");  // Liste içerisinde arama yap.  **if** (list.Contains("Mercedes"))  Console.WriteLine("OK."); // OK  **int** index = list.IndexOf("BMW");  Console.WriteLine(index); // 0  // 2. pozisyondaki nesne  Console.WriteLine(list[1]);  // Mercedes | ' Yeni bir ArrayList nesnesi ' oluştur. **Dim** list **As** ArrayList  list = **New** ArrayList()  ' Listeye yeni nesneler ekle.  list.Add("Audi")  list.Add("Mercedes")  ' Liste boyutunu yazdır.  Console.WriteLine(list.Count) ' 2  ' Liste içerisinden nesneyi sil  list.Remove("Audi")  ' 0 pozisyonuna yeni nesne ekle. list.Insert(0, "BMW")  ' Liste içerisinde arama yap.  **If** list.Contains("Mercedes") **Then**  Console.WriteLine("OK.") ' OK  **End** **If**  **Dim** index **As** **Integer**  index = list.IndexOf("BMW")  Console.WriteLine(index) ' 0  ' 2. pozisyondaki nesne  Console.WriteLine(list(1))  ' Mercedes |

Kod 161, IList Örnek Kullanımı

Koleksiyonlarda indeks değerinin 0'dan başladığına dikkat edilmelidir.

Sözlükler

Sözlükler, anahtar – değer ikililerinin yönetimi için kullanılması gereken yapılardır. Örneğin il trafik kodu – il adı, T.C. Kimlik Numarası – kişi Adı sözlük kullanımına örnek olarak verilebilir.

Sözlük yapısı IDictionary arabirimi ile ifade edilmektedir. Listeler ile sözlükler arasındaki belirgin farklar aşağıda listelenmiştir.

1. Liste kullanımında temel amaç farklı nesneleri saklamak ve nesnelere indeksler yardımıyla erişmektir. Sözlük kullanımı ile hedeflenen anahtar – değer ikililerini saklamak ve anahtar yardımıyla nesneye erişimdir.
2. Sözlük içerisinde anahtar değerleri tekil olmalıdır. Listeler içerisinde aynı nesne farklı pozisyonlarda bulunabilir.
3. Sözlük içerisinde bulunan anahtar – değer ikililerine indeks yardımıyla erişilememektedir.

IDictionary tanımı aşağıda gösterilmiştir.

C#

**public** **interface** IDictionary : ICollection, IEnumerable {

**void** Add(**object** key, **object** value);

**void** Clear();

**bool** Contains(**object** key);

IDictionaryEnumerator GetEnumerator();

**void** Remove(**object** key);

**bool** IsFixedSize { **get**; }

**bool** IsReadOnly { **get**; }

**object** **this**[**object** key] { **get**; **set**; }

ICollection Keys { **get**; }

ICollection Values { **get**; }

}

VB.NET

**Public** **Interface** IDictionary

**Implements** ICollection, IEnumerable

**Sub** Add(**ByVal** key **As** **Object**, **ByVal** value **As** **Object**)

**Sub** Clear()

**Function** Contains(**ByVal** key **As** **Object**) **As** **Boolean**

**Function** GetEnumerator() **As** IDictionaryEnumerator

**Sub** Remove(**ByVal** key **As** **Object**)

**ReadOnly** **Property** IsFixedSize **As** **Boolean**

**ReadOnly** **Property** IsReadOnly **As** **Boolean**

**Property** Item(**ByVal** key **As** **Object**) **As** **Object**

**ReadOnly** **Property** Keys **As** ICollection

**ReadOnly** **Property** Values **As** ICollection

**End** **Interface**

Kod 162, IDictionary Arabirim Tanımı

Tanımdan da görüleceği üzere, IDictionary.GetEnumerator metodu, listelerden farklı olarak IDictionaryEnumerator döndürmektedir.

C#

**public** **interface** IDictionaryEnumerator : IEnumerator {

DictionaryEntry Entry { get; }

**object** Key { get; }

**object** Value { get; }

}

VB.NET

**Public** **Interface** IDictionaryEnumerator

**Implements** IEnumerator

**ReadOnly** **Property** Entry **As** DictionaryEntry

**ReadOnly** **Property** Key **As** **Object**

**ReadOnly** **Property** Value **As** **Object**

**End** **Interface**

Kod 163, IDictionaryEnumerator Arabirim Tanımı

IDictionaryEntry arabirimi, sözlük içerisinde bulunan anahtar – değer (sırasıyla Key, Value) ikililerine erişim için kullanılmaktadır.

DictionaryEntry isimli yapı (struct), anahtar ve değer alanlarını içermektedir.

IDictionary arabirim üye bilgileri aşağıda gösterilmiştir.

| Üye Adı | Açıklama |
| --- | --- |
| IsFixedSize | Koleksiyonun tekrar boyutlandırılabilirliğini belirtmektedir. Değerinin true olması koleksiyon boyutunun sabit olduğunu belirtmektedir. |
| IsReadOnly | Koleksiyonun salt okunur olup olmadığını döndürmektedir. |
| Item | Koleksiyon içerisindeki herhangi bir nesneye erişebilmeyi sağlayan metottur. |
| Keys | Koleksiyon içerisindeki anahtarları ICollection olarak döndüren özelliktir. |
| Values | Koleksiyon içerisindeki değerleri ICollection olarak döndüren özelliktir. |
| Add | Anahtar – değer ikilisi eklemek için kullanılması gereken metottur. |
| Clear | Koleksiyon içeriğini temizlemek amacıyla kullanılması gereken metottur. |
| Remove | Parametre olarak iletilen anahtar değere karşılık gelen nesneyi koleksiyondan çıkarmak amacıyla kullanılmalıdır. |

.NET sınıf kütüphanesinin bir parçası olarak System.Collections ad alanı içerisinde tanımlı HashTable sınıfı IDictionary gerçekleştirimini yapmaktadır.

IDictionary arabirimi üyelerinin örnek kullanımı aşağıda gösterilmiştir.

C#

Hashtable iller = **new** Hashtable();

// Anahtar - değer ikililerini ekle.

iller.Add(51, "Niğde");

iller.Add(1, "Adana");

iller.Add(6, "Ankara");

**object** o = iller[51]; // Niğde

// 6 değerine sahip anahtarı koleksiyondan sil.

iller.Remove(6); // Ankara

**foreach** (DictionaryEntry e **in** iller)

Console.WriteLine("{0} - {1}", e.Key, e.Value);

VB.NET

**Dim** iller **As** HashTable = **New** Hashtable()

' Anahtar - değer ikililerini ekle.

iller.Add(51, "Niğde")

iller.Add(1, "Adana")

iller.Add(6, "Ankara")

**Dim** o **As** **Object** = iller(51) ' Niğde

' 6 değerine sahip anahtarı koleksiyondan sil.

iller.Remove(6)

**For** **Each** e **As** DictionaryEntry **In** iller

Console.WriteLine("{0} - {1}", e.Key, e.Value)

**Next**

Kod 164, IDictionary Sınıfı Örnek Kullanımı

Örnek çıktısı aşağıda gösterilmiştir.

51 - Niğde

1 - Adana

Koleksiyon içerisinde bulunan her bir nesneye DictionaryEntry ile erişildiğine dikkat ediniz.

Jenerik Mimari

Koleksiyon mimarisi, .NET 2.0 sürümünden itibaren jenerik tiplerin CLR tarafından desteklenmesi ile birlikte, jenerikleri de içerecek şekilde düzenlenmiştir.

Bu kapsamda, System.Collections.Generic ad alanı içerisinde, mimari yapı aynı kalmakla birlikte jenerik arabirimler tanımlanmıştır.

| Arabirim | Jenerik Sürümü |
| --- | --- |
| IEnumerator | IEnumerator<T> |
| IEnumerable | IEnumerable<T> |
| ICollection | ICollection<T> |
| IList | IList<T> |
| IDictionary | IDictionary<T> |

Jenerik koleksiyon mimarisi, jenerik olmayan mimariyi miras alır şekilde tanımlanmıştır.

C#

**public** **interface** IList<T> :ICollection<T>, IEnumerable<T>, IEnumerable {

**int** IndexOf(T item);

**void** Insert(**int** index, T item);

**void** RemoveAt(**int** index);

T **this**[**int** index] { **get**; **set**; }

}

VB.NET

**Public** **Interface** IList(**Of** T)

**Implements** ICollection(**Of** T), IEnumerable(**Of** T), IEnumerable

**Function** IndexOf(**ByVal** item **As** T) **As** **Integer**

**Sub** Insert(**ByVal** index **As** **Integer**, **ByVal** item **As** T)

**Sub** RemoveAt(**ByVal** index **As** **Integer**)

**Property** Item(**ByVal** index **As** **Integer**) **As** T

**End** **Interface**

Kod 165, Jenerik IList Arabirim Tanımı

System.Collections.Generic ad alanı içerisinde tanımlı List<T> (VB.NET için List(Of T)) örnek kullanımı aşağıda gösterilmiştir.

| C# | VB.NET |
| --- | --- |
| List<**string**> sList;  sList = **new** List<**string**>();  sList.Add("Audi"); | **Dim** sList **As** List(**Of** **String**)  sList = **New** List(**Of** **String**)  sList.Add("Audi") |

Kod 166, Örnek Jenerik List Sınıfı Kullanımı

Koleksiyon Sınıfları

.NET sınıf kütüphanesi içerisinde, bir önceki konuda aktarılan koleksiyon mimarisi kapsamında geliştirilen, kullanıma hazır bir çok koleksiyon sınıfı bulunmaktadır.

Bu sınıflar, aşağıda isim ve açıklamaları gösterilen ad alanları içerisinde organize edilmiştir.

| Ad Alanı | Açıklama |
| --- | --- |
| System.Collections | Temel koleksiyon arabirim ve sınıflarının bulunduğu ad alanıdır. |
| System.Collections.Specialized | Özelleştirilmiş koleksiyon sınıflarının bulunduğu ad alanıdır. |
| System.Collections.Generic | Jenerik koleksiyon arabirim ve sınıflarının bulunduğu ad alanıdır. |
| System.Collections.ObjectModel | Direkt veya miras alınarak kullanılabilecek jenerik koleksiyon sınıflarının bulunduğu ad alanıdır. |

Tablo 62, Koleksiyon Ad Alanları ve Açıklamaları

Jenerik Olmayan Koleksiyonlar

Programcılar, .NET sınıf kütüphanesi içerisinde sunulan kullanıma hazır bir çok koleksiyon sınıfı içerisinden amaçlarına en uygun olan sınıfı kullanmalıdır.

Jenerik olmayan temel koleksiyon sınıfları ve açıklamaları aşağıdadır.

| Sınıf | Açıklama |
| --- | --- |
| ArrayList | Farklı nesnelerin içerisinde saklanabileceği, tekrar boyutlandırılabilir liste koleksiyonudur. |
| BitArray | Mantıksal (Boolean) değerlerin içerisinde saklanabileceği liste koleksiyonudur. |
| CollectionBase | Programcıların miras alarak farklı liste koleksiyon sınıfları tanımlayabilecekleri soyut sınıftır. |
| DictionaryBase | Programcıların miras alarak farklı sözlük koleksiyon sınıfları tanımlayabilecekleri soyut sınıftır. |
| HashTable | Anahtar – değer ikili nesnelerinin içerisinde saklanabileceği sözlük koleksiyon sınıfıdır. |
| Queue | Kuyruk sınıfıdır. |
| ReadOnlyCollectionBase | Programcıların miras alarak salt okunur koleksiyon sınıfları tanımlayabilecekleri soyut sınıftır |
| SortedList | Anahtar – değer ikili nesnelerinin içerisinde saklanabileceği ve sıralama desteği olan sözlük koleksiyon sınıfıdır |
| Stack | Yığın sınıfıdır. |

Tablo 63, Jenerik Olmayan Temel Koleksiyon Sınıfları ve Açıklamaları

System.Collections.Specialized ad alanı içerisinde farklı amaçlar için kullanılabilecek sınıflar bulunmaktadır.

| Sınıf | Açıklama |
| --- | --- |
| StringCollection | İçerisinde karakter dizi nesnelerin saklanabileceği koleksiyon sınıfıdır. |
| StringDictionary | İçerisinde karakter dizi tipinde olan anahtar – değer ikililerinin saklanabileceği sözlük koleksiyon sınıfıdır. |
| ListDictionary | Az sayıda anahtar – değer ikilisinin en performanslı şekilde yönetilebileceği sözlük koleksiyon sınıfıdır. |
| HybridDictionary | Az sayıda anahtar – değer ikilisi için ListDictionary, sayının artması durumunda HashTable kullanan sözlük koleksiyon sınıfıdır. |
| OrderedDictionary | İçerik yönetimini sıralı olarak yapabilmek amacıyla kullanılması gereken sözlük koleksiyon sınıfıdır. |
| NameValueCollection | İçerisinde karakter dizi tipinde olan anahtar – değer ikililerinin saklanabileceği sözlük koleksiyon sınıfıdır. |

Tablo 64, Jenerik Olmayan Temel Koleksiyon Sınıfları ve Açıklamaları

Koleksiyon içerisinde arama yapma, sıralama gibi işlevlerin arka planda sınıflar tarafından nasıl sağlandığının bilinmesi büyük önem taşımaktadır. Bu amaçla, koleksiyon sınıfları örnek kullanımları hakkında detaylı bilgi verilmeden önce, nesne eşitliği ve karşılaştırılabilme konuları öncelikli olarak aktarılacaktır.

Nesne Eşitliği ve Karşılaştırılabilme

Kitabın daha önceki bölümlerinde nesne eşitliği konusunda bilgiler verilmişti. Bu konu başlığında, nesne eşitliği hakkında özet bilgiler ve nesnelerin karşılaştırılabilirlik nitelikleri aktarılacaktır.

Nesne Eşitliği

Object.Equals metodu, nesnelerin eşitliğini kontrol etmek amacıyla kullanılan sanal metottur. Farklı sınıflar bu metodu örterek nesne eşitlik davranışını değiştirebilirler.

| C# | VB.NET |
| --- | --- |
| **class** Araba {  **private** **string** plaka;  **public** **string** Plaka {  **get** {  **return** plaka;  }  }  **public** Araba(**string** plaka) {  **this**.plaka = plaka;  }  } | **Class** Araba  **Private** \_Plaka **As** **String**  **Public** **ReadOnly** **Property** Plaka()   **As** **String**  **Get**  **Return** \_Plaka  **End** **Get**  **End** **Property**  **Public** **Sub** **New**(**ByVal** plaka \_  **As** **String**)  **Me**.\_Plaka = plaka  **End** **Sub**  **End** **Class** |

Yukarıdaki kod örneğinde, yapıcı metodu karakter dizi tipinde değer alan Araba isminde sınıf tanımı yapılmıştır.

| C# | VB.NET |
| --- | --- |
| Araba a1 = **new** Araba("06AA001");  Araba a2 = **new** Araba("06AA001");  Console.WriteLine(a1.Equals(a2));  Console.WriteLine(  **object**.Equals(a1, a2));  Console.WriteLine(  **object**.ReferenceEquals(a1, a2)); | **Dim** a1 **As** Araba = **New** Araba("06AA001")  **Dim** a2 **As** Araba = **New** Araba("06AA001")  Console.WriteLine(a1.Equals(a2))  Console.WriteLine(**Object**.Equals(a1, a2))  Console.WriteLine(  **Object**.ReferenceEquals(a1, a2)) |

Object.Equals çağrımı ile elde edilen çıktı aşağıdadır.

False  
False  
False

Miras alan sınıflar tarafından örtülmediği sürece, Object.Equals metodu nesnelerin referansları üzerinden eşitlik kontrolü yapmaktadır. Örnekte, a1 ve a2 isimli nesneler, aynı sınıfa ait ancak referansları farklı olan nesnelerdir.

Nesnelerin eşitlik davranışını, Object.Equals metodunu örterek değiştirmek mümkündür.

C#

**class** Araba {

. . .

**public** **override** **bool** Equals(**object** obj) {

Araba a = obj **as** Araba;

**if** (a == **null**)

**return** **false**;

**return** **this**.plaka == a.plaka;

}

}

VB.NET

**Class** Araba

. . .

**Public** **Overrides** **Function** Equals(**ByVal** obj **As** **Object**) **As** **Boolean**

**Dim** a **As** Araba = **TryCast**(obj, Araba)

**If** a **Is** **Nothing** **Then**

**Return** **False**

**End** **If**

**Return** **Me**.\_Plaka = a.\_Plaka

**End** **Function**

**End** **Class**

Aynı kodun çalıştırılması durumunda aşağıdaki çıktı üretilecektir.

True  
True  
False

Statik olan Object.ReferenceEquals metodunun False değerini döndürdüğüne dikkat ediniz. Plakası aynı olan araba nesnelerinin eşit olarak kabul edilmesi gerektiği Object.Equals metodunu örterek sağlansa da, her iki nesnenin referansının farklı olduğu, yani nesnelerin farklı olduğu unutulmamalıdır.

Nesne Özet Değeri

Object.GetHashCode metodu, nesneleri tekil olarak ifade edebilen tam sayı değeri döndüren metottur. Nesne eşitlik davranışının Equals metodu örtülerek değiştirilmesi durumunda, bu metodun da örtülmesi önerilmektedir.

C#

**class** Araba {

. . .

**public** **override** **int** GetHashCode() {

**return** plaka.GetHashCode();

}

}

VB.NET

**Class** Araba

. . .

**Public** **Overrides** **Function** GetHashCode() **As** **Integer**

**Return** \_Plaka.GetHashCode()

**End** **Function**

**End** **Class**

Nesnelerin özet bilgi değerleri, örneğin HashTable sözlük sınıfı tarafından kullanılmaktadır.

İngilizce karşılığı ile Hash, büyük miktar verinin özetini ifade etmektedir. Örneğin 10 MB boyutunda olan verinin özeti sadece 128 BYTE olabilir.  
  
Özet bilginin temel amacı, büyük boyutlarda olan verilerin daha küçük boyutlarda olan özet veri ile ifade edilebilmelerini sağlamaktır. Özet bilgi kullanımında anahtar nokta, büyük miktarda veride olabilecek herhangi bir değişiklikte özet bilgi değerinin de değişmesidir.

Karşılaştırılabilme

Karşılaştırılabilme, farklı nesnelerin birbirleri ile karşılaştırılabilirliğini ifade etmektedir. Örneğin sayısal tipler, karakter dizi değerler karşılaştırılabilen nesnelerdir.

System.IComparable arabirimi, karşılaştırma amacıyla kullanılan arabirimdir. Karşılaştırılabilmeyi destekleyen nesneler bu arabirim gerçekleştirimini yapmaktadır.

| C# | VB.NET |
| --- | --- |
| **public** **interface** IComparable {  **int** CompareTo(**object** obj);  } | **Public** **Interface** IComparable  **Function** CompareTo(**ByVal** obj   **As** **Object**)   **As** **Integer**  **End** **Interface** |

Kod 167, System.IComparable Arabirim Tanımı

IComparable.CompareTo metodunun 0 (sıfır) değerini döndürmesi, bu arabirim gerçekleştirimini yapan nesne ile obj parametresi ile iletilen nesnenin aynı değere sahip olduğunu ifade etmektedir.

Metodun pozitif döndürmesi, obj parametresi ile iletilen nesnenin daha küçük olduğunu, negatif döndürmesi daha büyük olduğunu belirtmektedir.

IComparable<T>, IComparable arabiriminin jenerik sürümüdür.

C#

**public** **struct** Int32 : IComparable<**int**>, ... {

**public** **int** CompareTo(**int** value) {

**if** (**this** < value) {

**return** -1;

}

**if** (**this** > value) {

**return** 1;

}

**return** 0;

}

. . .

}

VB.NET

**Public** **Structure** Int32

**Implements** IComparable(**Of** **Integer**),

. . .

**Public** **Function** CompareTo(**ByVal** value **As** **Integer**) **As** **Integer**

**If** (**Me** < value) **Then**

**Return** -1

**End** **If**

**If** (**Me** > value) **Then**

**Return** 1

**End** **If**

**Return** 0

**End** **Function**

. . .

**End** **Structure**

Yukarıdaki kod örneğinde, tam sayı tipi olan System.Int32 yapısına ait kaynak kodun bir bölümü gösterilmiştir.

Koleksiyon Arabirim Tanımları

System.Collections ad alanı içerisinde tanımlı olan IEqualityComparer ve IComparer isimli arabirimler, nesnelerin eşitlik ve karşılaştırılabilme niteliklerinin farklı nesneler aracılığı ile yönetilebilmesini sağlamaktadır.

Jenerik sürümleri olan IEqualityComparer<T> ve IComparer<T> arabirimleri System.Collections.Generic ad alanı içerisinde tanımlıdır.

C#

**public** **interface** IEqualityComparer {

**bool** Equals(**object** x, **object** y);

**int** GetHashCode(**object** obj);

}

**public** **interface** IEqualityComparer<T> {

**bool** Equals(T x, T y);

**int** GetHashCode(T obj);

}

**public** **interface** IComparer {

**int** Compare(**object** x, **object** y);

}

**public** **interface** IComparer<T> {

**int** Compare(T x, T y);

}

VB.NET

**Public** **Interface** IEqualityComparer

**Function** Equals(**ByVal** x **As** **Object**, **ByVal** y **As** **Object**) **As** **Boolean**

**Function** GetHashCode(**ByVal** obj **As** **Object**) **As** **Integer**

**End** **Interface**

**Public** **Interface** IEqualityComparer(**Of** T)

**Function** Equals(**ByVal** x **As** T, **ByVal** y **As** T) **As** **Boolean**

**Function** GetHashCode(**ByVal** obj **As** T) **As** **Integer**

**End** **Interface**

**Public** **Interface** IComparer

**Function** [Compare](**ByVal** x **As** **Object**, **ByVal** y **As** **Object**) **As** **Integer**

**End** **Interface**

**Public** **Interface** IComparer(**Of** T)

**Function** [Compare](**ByVal** x **As** T, **ByVal** y **As** T) **As** **Integer**

**End** **Interface**

Kod 168, IEqualityComparer ve IComparer Arabirim Tanımları

Koleksiyon Sınıfları Kullanım Bilgileri

Koleksiyon sınıfları, .NET koleksiyon mimarisi çerçevesinde, kitabın daha önceki konularında aktarılan koleksiyon arabirim gerçekleştirimlerini yapan sınıflardır.

Koleksiyon sınıflarının organizasyonu, kullanım amaçlarına uygun olarak yapılmıştır. Anlatımda, koleksiyon arabirimleri konusunda aktarılan bilgiler tekrar edilmeyecek, sınıfların arabirim gerçekleştirimleri ve üyeleri hakkında önemli görülen bilgiler sunulacaktır.

ArrayList Kullanımı

ArrayList sınıfı, farklı tipte nesneleri saklayabilen koleksiyon sınıfıdır.

| C# | VB.NET |
| --- | --- |
| **public** **class** ArrayList :  IList,  ICollection,  IEnumerable,  ICloneable {  . . .  } | **Public** **Class** ArrayList  **Implements** IList, \_   ICollection, \_  IEnumerable, \_   ICloneable  . . .  **End** **Class** |

Kod 169, ArrayList Sınıfı Tanımı

ArrayList sınıfı, IList, ICollection ve IEnumerable koleksiyon arabirim gerçekleştirimlerini yapmaktadır.

**Yapıcı Metot Bilgileri**

ArrayList nesneleri, sınıfa ait farklı yapıcı metotlar kullanılarak oluşturulabilir.

C#

// Parametre almayan yapıcı metot.

**public** ArrayList();

// Parametre olarak iletilen nesne içeriğini listeye kopyalayan yapıcı metot.

**public** ArrayList(ICollection c);

// İlk kapasite belirtilen yapıcı metot.

**public** ArrayList(**int** capacity);

VB.NET

' Parametre almayan yapıcı metot.

**Public** **Sub** **New**()

' Parametre olarak iletilen nesne içeriğini listeye kopyalayan yapıcı metot.

**Public** **Sub** **New**(**ByVal** c **As** ICollection)

' İlk kapasite belirtilen yapıcı metot.

**Public** **Sub** **New**(**ByVal** capacity **As** **Integer**)

Kod 170, ArrayList Yapıcı Metotları

Kapasite (Capacity) ve liste içerisindeki eleman sayısı (Count) birbirlerine karıştırılabilen konulardır.  
  
Kapasite, dizi içerisine eklenebilecek elemanlar için önceden yapılan hafıza ayrımını alanı ifade etmektedir. Örneğin dizi eleman sayısı 5 iken, kapasite 15 olabilir. Bu durumda, 10 elemanın daha eklenebilmesi için ArrayList nesnesi içerisinde gerekli hafıza ayrımının yapıldığı anlaşılmalıdır.  
  
Kapasite kullanımı ile hedeflenen performans kazanımı sağlamaktır. Yapıcı metoda tahmini kapasite bilgisi parametre olarak iletilerek veya ArrayList.Capacity özelliğine değer atanarak, listenin gerekli hafıza ayrımını önceden yapması, dolayısıyla listeye yapılan eleman ekleme işleminde, her bir eklenen eleman için hafıza ayırma işleminin tekrarlanması engellenmektedir.  
  
Kapasite konusu, farklı koleksiyon sınıfları için de geçerlidir.  
  
Not: Kapasite, eleman sayısından daha az olamaz.

Aşağıda yapıcı metot ve kapasite ile ilgili kod örneği bulunmaktadır.

| C# | VB.NET |
| --- | --- |
| ArrayList list1;  list1 = **new** ArrayList();  list1.Add("Mercedes");  list1.Add("Audi");  // list1 isimli nesne içeriğini // list2 isimli nesneye kopyalayan // yapıcı metot  ArrayList list2;  list2 = **new** ArrayList(list1);  // İlk kapasite belirtilen yapıcı // metot  ArrayList list3;  list3 = **new** ArrayList(5);  list2.Add("BMW");  // list2 nesnesi kapasitesi 4 // değerini almaktadır. ArrayList, // optimizasyon amacıyla eleman  // sayısına göre daha fazla // kapasiteye sahip olmaktadır.  Console.WriteLine(list2.Capacity);  // Liste içerisindeki eleman sayısı // 3 değerini döndürecektir.  Console.WriteLine(list2.Count);  list2.Capacity = 10;  // Kapasite artırılmış olmasına rağmen // liste eleman sayısı yine 3 // değerini alacaktır.  Console.WriteLine(list2.Count); | **Dim** list1 **As** ArrayList  list1 = **New** ArrayList()  list1.Add("Mercedes")  list1.Add("Audi")  ' list1 isimli nesne içeriğini ' list2 isimli nesneye kopyalayan ' yapıcı metot  **Dim** list2 **As** ArrayList  list2 = **New** ArrayList(list1)  ' İlk kapasite belirtilen yapıcı ' metot  **Dim** list3 **As** ArrayList  list3 = **New** ArrayList(5)  list2.Add("BMW")  ' list2 nesnesi kapasitesi 4 ' değerini almaktadır. ArrayList, ' optimizasyon amacıyla eleman  ' sayısına göre daha fazla ' kapasiteye sahip olmaktadır.  Console.WriteLine(list2.Capacity)  ' Liste içerisindeki eleman sayısı ' 3 değerini döndürecektir.  Console.WriteLine(list2.Count)  list2.Capacity = 10  ' Kapasite artırılmış olmasına rağmen ' liste eleman sayısı yine 3 ' değerini alacaktır.  Console.WriteLine(list2.Count) |

Kod 171, ArrayList Yapıcı Metot ve Kapasite Örnekleri

**Üye Bilgileri**

ArrayList sınıfına ait önemli üye bilgileri ve açıklamaları aşağıdadır.

| Üye Adı | Açıklama |
| --- | --- |
| Reverse | Liste içeriğini tersine çevirir. |
| Sort | Liste içeriğini sıralar. |
| ToArray | Liste içeriğini yeni bir dizi nesnesine kopyalar. |
| TrimToSize | Liste kapasitesini, liste eleman sayısına düşürür / eşitler. |

Tablo 65, ArrayList Üye Bilgileri

C#

ArrayList list1;

list = **new** ArrayList();

list1.Add("Mercedes");

list1.Add("Audi");

// Tersine çevir.  
// list1[0] = "Audi"  
// list1[1] = "Mercedes"

list1.Reverse();

**string** [] dizi;

dizi = (**string** [])list1.ToArray(**typeof**(**string** []));

VB.NET

**Dim** list1 **As** ArrayList

list1 = **New** ArrayList()

list1.Add("Mercedes")

list1.Add("Audi")

' Tersine çevir.  
' list1[0] = "Audi"  
' list1[1] = "Mercedes"

list1.Reverse()

**Dim** dizi() **As** **String**

dizi = **CType**(list1.ToArray(**GetType**(**String**)), **String**())

Kod 172, ArrayList Üyeleri Kullanım Örnekleri

**Sıralama**

ArrayList.Sort metodu, liste içerisindeki elemanları sıralamak için kullanılmalıdır.

Örnek sıralama aşağıda gösterilmiştir.

| C# | VB.NET |
| --- | --- |
| ArrayList list1;  list1 = **new** ArrayList();  list1.Add("Veli");  list1.Add("Hasan");  list1.Add("Cemal");  list1.Sort();  **foreach** (**string** s **in** list1)  Console.WriteLine(s);  // Cemal  // Hasan  // Veli | **Dim** list1 **As** ArrayList  list1 = **New** ArrayList()  list1.Add("Veli")  list1.Add("Hasan")  list1.Add("Cemal")  list1.Sort()  **Dim** s **As** **String**  **For** **Each** s **As** **String** **In** list1  Console.WriteLine(s)  **Next**  ' Cemal  ' Hasan  ' Veli |

Kod 173, Temel Array.Sort Metodu Kullanımı

Sıralama işleminin yapılabilmesi için liste içerisinde bulunan nesneler IComparable arabirim gerçekleştirimini yapmalıdır. Nesnelerin IComparable arabirim gerçekleştirimini yapmamış olmaları durumda Sort metodu istisna fırlatacaktır.

ArrayList.Sort metodu yüklenmiş sürümleri, IComparer arabirim gerçekleştirimini yapan herhangi bir nesneyi parametre olarak kabul edebilmektedir.

C#

**public** **virtual** **void** Sort(IComparer comparer);

**public** **virtual** **void** Sort(**int** index, **int** count, IComparer comparer);

VB.NET

**Public** **Overridable** **Sub** Sort(**ByVal** comparer **As** IComparer)

**Public** **Overridable** **Sub** Sort(**ByVal** index **As** **Integer**,   
 **ByVal** count **As** **Integer**,   
 **ByVal** comparer **As** IComparer)

Kod 174, ArrayList.Sort Metot Tanımları

Yüklenmiş sürümler iki temel işlevi sağlamak için aşağıda listelenen durumlara kullanılabilir.

1. Liste içerisinde bulunan nesneler IComparable arabirim gerçekleştirimini yapmamıştır.
2. Liste içerisinde bulunan nesneler IComparable arabirim gerçekleştirimini yapmalarına rağmen, sıralamanın farklı yöntemler ile yapılması istenmektedir.

Aşağıda örnek sınıf tanımı ve kullanım örneği gösterilmiştir.

C#

**class** Araba {

**private** **string** plaka;

**public** **string** Plaka {

**get** {

**return** plaka;

}

}

. . .

}

**class** ArabaComparer : IComparer {

**public** **int** Compare(**object** x, **object** y) {

Araba a1 = x **as** Araba;

Araba a2 = y **as** Araba;

**if** (a1 == **null** || a2 == **null**)

**throw** **new** Exception(  
 "Geçersiz işlem, nesneler araba tipinde olmalıdır.");

**return** String.Compare(a1.Plaka, a2.Plaka);

}

}

VB.NET

**Class** Araba

**Private** \_Plaka **As** **String**

**Public** **ReadOnly** **Property** Plaka() **As** **String**

**Get**

**Return** \_Plaka

**End** **Get**

**End** **Property**

. . .

**End** **Class**

**Class** ArabaComparer

**Implements** IComparer

**Public** **Function** Compare(**ByVal** x **As** **Object**, **ByVal** y **As** **Object**) **As** **Integer**   
 **Implements** IComparer.Compare

**Dim** a1 **As** Araba = **TryCast**(x, Araba)

**Dim** a2 **As** Araba = **TryCast**(y, Araba)

**If** a1 **Is** **Nothing** **Or** a2 **Is** **Nothing** **Then**

**Throw** **New** Exception(  
 "Geçersiz işlem, nesneler araba tipinde olmalıdır.")

**End** **If**

**Return** **String**.Compare(a1.Plaka, a2.Plaka)

**End** **Function**

**End** **Class**

Kod 175, ArrayList.Sort ve IComparer Kullanımı

BitArray Kullanımı

BitArray sınıfı, mantıksal değerler yardımı ile ikil işlemlerin kolaylıkla yapılabilmesini sağlayan koleksiyon sınıfıdır.

Mantıksal True, ikil 1 değerini, Mantıksal False, ikil 0 değerini ifade etmektedir.

| C# | VB.NET |
| --- | --- |
| **public** **sealed** **class** BitArray :   ICollection,   IEnumerable,   ICloneable {  . . .  } | **Public** **NotInheritable** **Class** BitArray  **Implements** ICollection,   IEnumerable,   ICloneable  . . .  **End** **Class** |

Kod 176, BitArray Sınıf Tanımı

BitArray sınıfının, eleman eklemek ve liste içerisinden herhangi bir elemanı silmek için Add ve Remove metotları bulunmamaktadır. Listenin boyutu, yapıcı metoda iletilen boyut parametre değeri ile belirlenmekte, daha sonra BitArray.Length özelliğine istenilen tam sayı değeri atanarak liste boyutu düzenlenebilmektedir.

| C# | VB.NET |
| --- | --- |
| BitArray bits = **new** BitArray(3);  bits[0] = **true**;  bits[1] = **false**;  bits[2] = **false**;  bits.Length = 5;  bits[3] = **false**;  bits[4] = **true**;  **foreach** (**bool** b **in** bits)  Console.WriteLine(b); | **Dim** bits **As** BitArray = **New** BitArray(3)  bits(0) = **True**  bits(1) = **False**  bits(2) = **False**  bits.Length = 5  bits(3) = **False**  bits(4) = **True**  **For** **Each** b **As** **Boolean** **In** bits  Console.WriteLine(b)  **Next** |

Kod 177, BitArray Sınıfı Boyutunun Belirlenmesi

Yukarıdaki örnekte bits isimli nesne oluşturulmakta ve ilk boyutu 3 olarak belirlenmektedir. Bu durumda ön tanımlı olarak liste içerisinde 3 adet False (ikil 0) değerine sahip eleman bulunmaktadır.

BitArray.Length özelliğine 5 değerinin atanması ile liste boyutu artırılmaktadır.

Örnek kod çıktısı aşağıda gösterilmiştir.

True

False

False

False

True

**Üye Bilgileri**

BitArray sınıfının temel kullanım amacı, listenin sahip olduğu elemanlar üzerinde ikil işlemlerin kolaylıkla yapılabilmesini sağlamaktır.

İkil işlemler (ing. Binary - Bitwise Operations) konusunda kitabın daha önceki bölümlerinde gerekli bilgiler bulunmaktadır.

BitArray üye bilgileri ve açıklamaları aşağıda gösterilmiştir.

| Üye Adı | Açıklama |
| --- | --- |
| And | Parametre olarak iletilen BitArray nesnesi ile ikil VE işlemini yapar ve işlem sonucunu içeren yeni bir BitArray nesnesi döndürür. |
| Or | Parametre olarak iletilen BitArray nesnesi ile ikil VEYA işlemini yapar ve işlem sonucunu içeren yeni bir BitArray nesnesi döndürür. |
| Xor | Parametre olarak iletilen BitArray nesnesi ile ikil XOR işlemini yapar ve işlem sonucunu içeren yeni bir BitArray nesnesi döndürür. |
| Not | BitArray nesnesi elemanları üzerinde ikil DEĞİL işlemini yapar. |
| SetAll | Tüm elemanların parametre olarak iletilen mantıksal değeri almalarını sağlar. |

Tablo 66, BitArray Sınıfı Üye Bilgileri

Üye bilgileri örnek kullanımları ve açıklamalar için aşağıdaki örnek incelenmelidir.

| C# | VB.NET |
| --- | --- |
| **static** **void** BitArrayYaz(BitArray arr) {  **foreach** (**bool** b **in** arr)  Console.Write(b ? "1" : "0");  Console.WriteLine();  }  BitArray bits1 = **new** BitArray(3);  bits1[0] = **true**;  bits1[1] = **false**;  bits1[2] = **false**;  BitArray bits2 = **new** BitArray(3);  bits2[0] = **true**;  bits2[1] = **true**;  bits2[2] = **false**;  BitArrayYaz(bits1.And(bits2));  // 100  BitArrayYaz(bits1.Or(bits2));  // 110  BitArrayYaz(bits1.Xor(bits1));  // 000  bits1.SetAll(**true**);  bits2.Not();  BitArrayYaz(bits1.And(bits2));  // 001 | **Shared** **Sub** BitArrayYaz(**ByVal** arr   **As** BitArray)  **For** **Each** b **As** **Boolean** **In** arr  Console.Write(Iff(b, "1", "0"))  **Next**  Console.WriteLine()  **End** **Sub**  **Dim** bits1 **As** BitArray  bits1 = **New** BitArray(3)  bits1(0) = **True**  bits1(1) = **False**  bits1(2) = **False**  **Dim** bits2 **As** BitArray  bits2 = **New** BitArray(3)  bits2(0) = **True**  bits2(1) = **True**  bits2(2) = **False**  BitArrayYaz(bits1.**And**(bits2))  ' 100  BitArrayYaz(bits1.**Or**(bits2))  ' 110  BitArrayYaz(bits1.**Xor**(bits1))  ' 000  bits1.SetAll(**True**)  bits2.**Not**()  BitArrayYaz(bits1.**And**(bits2))  ' 001 |

Kod 178, BitArray Kullanım Örnekleri

System.Collections.BitVector32 sınıfı, BitArray sınıfından farklı olarak, tam sayılar üzerinde ikil işlemler yapmayı kolaylaştıran metot ve özelliklere sahiptir.

HashTable Kullanımı

HashTable sınıfı, anahtar – değer ikili nesnelerinin içerisinde saklanabileceği sözlük koleksiyon sınıfıdır.

| C# | VB.NET |
| --- | --- |
| **public** **class** Hashtable :  IDictionary,  ICollection,  IEnumerable,  ISerializable,  IDeserializationCallback,  ICloneable {  . . .  } | **Public** **Class** Hashtable  **Implements** IDictionary,  ICollection,  IEnumerable,  ISerializable,   IDeserializationCallback,  ICloneable  . . .  **End** **Class** |

Kod 179, HashTable Sınıf Tanımı

HashTable sınıfın temel amacı, belirtilen anahtara karşılık liste içerisinde bulunan değeri elde etmektir. Anahtar, herhangi bir nesne olabilmektedir. Anahtar değerlerin eşitliği, kitabın bir önceki konusunda aktarılan nesne eşitliği kuralları ile yönetilmektedir.

**Yapıcı Metot Bilgileri**

HashTable, yaklaşık 15 adet yüklenmiş yapıcı metot tanımlamaktadır. Aşağıda en sık kullanılan bazı yapıcı metotlar ve açıklamaları sunulmuştur.

C#

// Parametresiz yapıcı metot.

**public** Hashtable();

// d isimli parametre iletilen sözlük koleksiyon içeriğini kopyar.

**public** Hashtable(IDictionary d);

// Eşitlik kontrolünde kullanılması istenilen IEqualityComparer nesnesinin  
// iletimini sağlar.

**public** Hashtable(IEqualityComparer equalityComparer);

// Ön tanımlı kapasite değerini belirtir.

**public** Hashtable(**int** capacity);

VB.NET

' Parametresiz yapıcı metot.

**Public** **Sub** **New**()

' d isimli parametre iletilen sözlük koleksiyon içeriğini kopyar.

**Public** **Sub** **New**(**ByVal** d **As** IDictionary)

' Eşitlik kontrolünde kullanılması istenilen IEqualityComparer nesnesinin  
' iletimini sağlar.

**Public** **Sub** **New**(**ByVal** equalityComparer **As** IEqualityComparer)

' Ön tanımlı kapasite değerini belirtir.

**Public** **Sub** **New**(**ByVal** capacity **As** **Integer**)

Kod 180, HashTable Yapıcı Metot Tanım ve Açıklamaları

HashTable kullanım örnekleri aşağıda gösterilmiştir.

| C# | VB.NET |
| --- | --- |
| Hashtable table = **new** Hashtable();  table.Add("Niğde", 51);  // Aşağıdaki gibi de kullanılabilir. // table["Niğde"] = 51; // Bu durumda anahtar yoksa ekleme // işlemi yapılmakta, varsa anahtara // karşılık gelen yeni değer  // yazılmaktadır.  table.Add("Ankara", 6);  table.Add("Konya", 42);  **int** i = (**int**)table["Niğde"];  Console.WriteLine(i);  // 51 değeri yazılacaktır.  **foreach** (DictionaryEntry e **in** table) {  Console.Write(e.Key);  Console.WriteLine(e.Value);  }  // Ankara6  // Konya42  // Niğde51 | **Dim** table **As** Hashtable  table = **New** Hashtable()  table.Add("Niğde", 51)  ' Aşağıdaki gibi de kullanılabilir. ' table("Niğde") = 51; ' Bu durumda anahtar yoksa ekleme ' işlemi yapılmakta, varsa anahtara ' karşılık gelen yeni değer  ' yazılmaktadır.  table.Add("Ankara", 6)  table.Add("Konya", 42)  **Dim** i **As** **Integer**  i = **CType**(table("Niğde"), **Integer**)  Console.WriteLine(i)  ' 51 değeri yazılacaktır.  **Dim** e **As** DictionaryEntry  **For** **Each** e **In** table  Console.Write(e.Key)  Console.WriteLine(e.Value)  **Next**  ' Ankara6  ' Konya42  ' Niğde51 |

Kod 181, Temel HashTable Kullanım Örnekleri

HashTable içerisindeki her elemanın DictionaryEntry nesneleri şeklinde saklandığına dikkat edilmelidir.

HashTable kullanımında en sık yapılabilecek hatalardan birisi, indeks kullanılarak HashTable içeriğine erişmeye çalışmaktır.   
  
Yukarıdaki kod örneği değerlendirildiğinde table[0] ifadesi sözlük içerisindeki ilk elemanı değil, null (VB.NET için Nothing) değerini döndürecektir. IList arabirim gerçekleştirimini yapan sınıflar ile sözlük sınıfları arasındaki fark, benzer ifadenin, IList arabirimi için liste içerisindeki pozisyona göre elemana erişimi sağlaması, IDictionary arabirimi için anahtar değeri döndürmesidir.  
  
HashTable kullanımında dikkat edilmesi gereken bir diğer konu, sözlük içerisindeki elemanların otomatik olarak **özet bilgi değerlerine** göre sıralanmış olduğudur. Yukarıdaki örnekte foreach döngüsü ile elde edilen çıktı ve sözlüğe elemanların eklenme sırası arasındaki farka dikkat ediniz.

**Üye Bilgileri**

HashTable üye bilgileri ve açıklamaları aşağıda gösterilmiştir.

| Üye Adı | Açıklama |
| --- | --- |
| ContainsKey | Parametre olarak iletilen anahtar nesnenin sözlük içerisinde bulunup bulunmadığını döndürür. |
| ContainsValue | Parametre olarak iletilen nesnenin, sözlük içerisinde bulunup bulunmadığını döndürür. |

Tablo 67, System.Collections.HashTable Sınıfı Üye Bilgileri

SortedList Kullanımı

SortedList sınıfı, liste sınıflarında olduğu gibi içeriğe pozisyonu ile erişim ve otomatik sıralama desteği bulunan sözlük sınıfıdır.

| C# | VB.NET |
| --- | --- |
| **public** **class** SortedList :  IDictionary,  ICollection,  IEnumerable,  ICloneable {  . . .  } | **Public** **Class** SortedList  **Implements** IDictionary,  ICollection,  IEnumerable,  ICloneable  . . .  **End** **Class** |

Kod 182, System.Collections.SortedList Sınıf Tanımı

**Yapıcı Metot Bilgileri**

SortedList sınıfına ait yapıcı metot tanım ve açıklamaları aşağıda gösterilmiştir.

C#

// Parametresiz yapıcı metot.

**public** SortedList();

// Sıralama için kullanılması istenilen ve IComparer arabirim   
// gerçekleştirimini yapan nesnenin iletildiği yapıcı metot.

**public** SortedList(IComparer comparer);

// d isimli parametre iletilen sözlük koleksiyon içeriğini kopyar.

**public** SortedList(IDictionary d);

// Ön tanımlı kapasite değerini belirtir.

**public** SortedList(**int** initialCapacity);

VB.NET

' Parametresiz yapıcı metot.

**Public** **Sub** **New**()

' Sıralama için kullanılması istenilen ve IComparer arabirim   
' gerçekleştirimini yapan nesnenin iletildiği yapıcı metot.

**Public** **Sub** **New**(**ByVal** comparer **As** IComparer)

**'** d isimli parametre iletilen sözlük koleksiyon içeriğini kopyar.

**Public** **Sub** **New**(**ByVal** d **As** IDictionary)

' Ön tanımlı kapasite değerini belirtir.

**Public** **Sub** **New**(**ByVal** initialCapacity **As** **Integer**)

Kod 183, System.Collections.SortedList Yapıcı Metot Tanım ve Açıklamaları

HashTable ve SortedList arasındaki fark aşağıdaki örnek ile gösterilmektedir.

C#

**enum** SozlukTipi {

HashTable,

SortedList

}

**static** IDictionary Olustur(SozlukTipi tip) {

IDictionary sonuc;

**if** (tip == SozlukTipi.HashTable)

sonuc = **new** Hashtable();

**else**

sonuc = **new** SortedList();

sonuc.Add("Iki", 2);

sonuc.Add("Üç", 3);

sonuc.Add("Bir", 1);

**return** sonuc;

}

**static** **void** Yaz(IDictionary d) {

**foreach** (DictionaryEntry e **in** d) {

Console.Write(e.Key);

Console.WriteLine(e.Value);

}

}

IDictionary hashTable = Olustur(SozlukTipi.HashTable);

IDictionary sortedList = Olustur(SozlukTipi.SortedList);

Yaz(hashTable);

Console.WriteLine("------------");

Yaz(sortedList);

VB.NET

**Enum** SozlukTipi

HashTable

SortedList

**End** **Enum**

**Shared** **Function** Olustur(**ByVal** tip **As** SozlukTipi) **As** IDictionary

**Dim** sonuc **As** IDictionary

**If** tip = SozlukTipi.HashTable **Then**

sonuc = **New** Hashtable()

**Else**

sonuc = **New** SortedList()

**End** **If**

sonuc.Add("Iki", 2)

sonuc.Add("Üç", 3)

sonuc.Add("Bir", 1)

**Return** sonuc

**End** **Function**

**Shared** **Sub** Yaz(**ByVal** d **As** IDictionary)

**For** **Each** e **As** DictionaryEntry **In** d

Console.Write(e.Key)

Console.WriteLine(e.Value)

**Next**

**End** **Sub**

**Dim** hashTable **As** IDictionary = Olustur(SozlukTipi.HashTable)

**Dim** sortedList **As** IDictionary = Olustur(SozlukTipi.SortedList)

Yaz(hashTable)

Console.WriteLine("------------")

Yaz(sortedList)

Kod 184, System.Collections.SortedList Kullanımı

Örnek kod parçasında, Olustur isimli metot, kendisine iletilen parametre değerine göre HashTable veya SortedList nesneleri oluşturmaktadır. Metodun dönüş değer tipinin IDictionary arabirimi olduğuna dikkat edilmelidir.

Kod çıktısı aşağıda gösterilmiştir.

Bir1

Üç3

Iki2

------------

Bir1

Iki2

Üç3

HashTable sınıfı tarafından yapılan sıralama elemanların özet değerlerine göre yapılmış olmasına rağmen, SortedList tarafından yapılan sıralama IComparable arabirimine göre yapılmıştır.

**Üye Bilgileri**

SortedList temel üye bilgileri ve açıklamaları aşağıda listelenmiştir.

| Üye Adı | Açıklama |
| --- | --- |
| GetByIndex | Parametre olarak iletilen pozisyon bilgisi ile eleman değerine erişimi sağlar. |
| GetKey | Parametre olarak iletilen pozisyon bilgisi ile eleman anahtarına erişimi sağlar. |
| IndexOfKey | Parametre olarak iletilen anahtar değerin liste içerisindeki pozisyonunu döndürür. |
| IndexOfValue | Parametre olarak iletilen nesnenin, liste içerisindeki pozisyonunu döndürür |
| RemoveAt | Parametre olarak iletilen pozisyonda bulunan elemanı listeden siler. |
| SetByIndex | Parametre olarak iletilen pozisyon ve nesne ile var olan eleman değerini değiştirir. |

Tablo 68, System.Collections.SortedList Sınıfı Üye Bilgileri

| C# | VB.NET |
| --- | --- |
| SortedList list = **new** SortedList();  list.Add("Sekiz", 8);  list.Add("Bir", 1);  list.Add("Dört", 4);  list.Add("Üç", 3);  // Sıralanmış olarak  // Bir, Dört , Sekiz, Üç  **object** o = list.GetByIndex(2);  // o, 8 değerini alacaktır.  o = list.GetKey(2);  // o, "Sekiz" değerini alacaktır.  // İlk pozisyondaki içeriği sil.  list.RemoveAt(0); | **Dim** list **As** SortedList  list = **New** SortedList()  list.Add("Sekiz", 8)  list.Add("Bir", 1)  list.Add("Dört", 4)  list.Add("Üç", 3)  ' Sıralanmış olarak  ' Bir, Dört , Sekiz, Üç  **Dim** o **As** **Object**  o = list.GetByIndex(2)  ' o, 8 değerini alacaktır.  o = list.GetKey(2)  ' o, "Sekiz" değerini alacaktır.  ' İlk pozisyondaki içeriği sil.  list.RemoveAt(0) |

Kod 185, System.Collections.SortedList Kullanım Örnekleri

OrderedDictionary Kullanımı

OrderedDictionary sınıfı, içerisine eklenen anahtar – değer ikililerine, eklendiği sıra / pozisyon ile erişim amacıyla kullanılması gereken sınıftır.

Bilindiği gibi HashTable sınıfı, pozisyon ile elemanlara erişimi desteklememekte, içeriğe erişim IEnumerable arabirimi ve foreach döngüsü ile yapılabilmektedir. SortedList sınıfı bu kısıtları farklı metotlar sağlayarak ortadan kaldırmasına rağmen, listeye eklenen elemanlar otomatik olarak sıralanmaktadır.

OrderedDictionary sınıfı;

1. Sözlük içeriğine pozisyon ile erişimin gerektiği.
2. Sözlük içerisinde bulunan elemanlara eklendiği sırayla erişimin gerektiği

Durumlarda kullanılmalıdır.

C#

**public** **class** OrderedDictionary :

IOrderedDictionary,

IDictionary,

ICollection,

IEnumerable,

ISerializable,

IDeserializationCallback {

. . .

}

**public** **interface** IOrderedDictionary :

IDictionary,

ICollection,

IEnumerable {

IDictionaryEnumerator GetEnumerator();

**void** Insert(**int** index, **object** key, **object** value);

**void** RemoveAt(**int** index);

**object** **this**[**int** index] { get; set; }

}

VB.NET

**Public** **Class** OrderedDictionary

**Implements** IOrderedDictionary,   
 IDictionary,   
 ICollection,   
 IEnumerable,   
 ISerializable,   
 IDeserializationCallback

. . .

**End Class**

**Public** **Interface** IOrderedDictionary

**Implements** IDictionary,   
 ICollection,   
 IEnumerable

**Function** GetEnumerator() **As** IDictionaryEnumerator

**Sub** Insert(**ByVal** index **As** **Integer**,   
 **ByVal** key **As** **Object**,   
 **ByVal** value **As** **Object**)

**Sub** RemoveAt(**ByVal** index **As** **Integer**)

**Default** **Property** Item(**ByVal** index **As** **Integer**) **As** **Object**

**End** **Interface**

Kod 186, System.Collections.Specialized.OrderedDictionary Sınıfı Tanımı

OrderedDictionary kullanımı ve metotları HashTable ve SortedList sınıfları ile büyük ölçüde benzerlik göstermektedir.

Aşağıdaki kod parçasında OrderedDictionary kullanımı örneklenmiştir.

| C# | VB.NET |
| --- | --- |
| OrderedDictionary list;  list = **new** OrderedDictionary();  list.Add("Sekiz", 8);  list.Add("Bir", 1);  list.Add("Dört", 4);  list.Add("Üç", 3);  // Herhangi bir sıralama  // yapılmamıştır, içerik:  // Sekiz, Bir, Dört, Üç  **object** o = list[0];  // Sözlük içerisinde ilk pozisyonda // bulunan elemana erişilmektedir. // o, 8 değerini alacaktır.  o = list["Üç"];  // HashTable sınıfında olduğu gibi // anahtar ile elemana erişilmektedir. // o, 3 değerini alacaktır.  // Sözlük 3. Pozisyonuna yeni // eleman eklenmektedir.  list.Insert(3, "Beş", 5);  // Sözlük içerisinde bulunan ilk // eleman silinmektedir.  list.RemoveAt(0);  **foreach** (DictionaryEntry e **in** list) {  Console.Write(e.Key);  Console.WriteLine(e.Value);  }  // Bir1 // Dört4 // Beş5 // Üç3 | **Dim** list **As** OrderedDictionary  list = **New** OrderedDictionary()  list.Add("Sekiz", 8)  list.Add("Bir", 1)  list.Add("Dört", 4)  list.Add("Üç", 3)  ' Herhangi bir sıralama  ' yapılmamıştır, içerik:  ' Sekiz, Bir, Dört, Üç  **Dim** o **As** **Object** = list(0)  ' Sözlük içerisinde ilk pozisyonda ' bulunan elemana erişilmektedir. ' o, 8 değerini alacaktır.  o = list("Üç")  ' HashTable sınıfında olduğu gibi ' anahtar ile elemana erişilmektedir. ' o, 3 değerini alacaktır.  ' Sözlük 3. Pozisyonuna yeni ' eleman eklenmektedir.  list.Insert(3, "Beş", 5)  ' Sözlük içerisinde bulunan ilk ' eleman silinmektedir.  list.RemoveAt(0)  **For** **Each** e **As** DictionaryEntry **In** list  Console.Write(e.Key)  Console.WriteLine(e.Value)  **Next**  ' Bir1 ' Dört4 ' Beş5 ' Üç3 |

Kod 187, System.Collections.Specialized.OrderedDictionary Örnekleri

Diğer Sözlük Sınıfları

ListDictionary ve HybridDictionary sınıfları performans optimizasyonu sağlamak amacıyla kullanılabilecek sözlük sınıflarıdır.

HashTable sınıfı, çok yüksek sayıda elemanın yönetimi için olabildiğince optimize edilmiş olmasına rağmen, az sayıda eleman (10 ve altı) içermesi durumunda bir miktar yönetim maliyetini beraberinde getirmektedir.

ListDictionary sınıfı, sözlük içerisinde az sayıda eleman olması durumu için optimize edilmiştir. Programcılar, sözlük eleman sayısının 10'u aşmayacağından emin olduklarında performans optimizasyonu sağlamak amacıyla bu sınıfı kullanmalıdır.

HybridDictionary sınıfı, 10 ve altı eleman sayısı için yönetimi ListDictionary sınıfı ile yapan, daha fazla eleman durumunda yönetimi HashTable ile sürdüren sözlük sınıfıdır. HybridDictionary sınıfı, listeye eklenebilecek eleman sayısının belirsiz olduğunda kullanılmalıdır.

Sözlük boyutunun limitlerin her zaman üzerinde olacağı varsayılan durumlarda HashTable sınıfı kullanımı tercih edilmelidir.

Queue Kullanımı

Queue sınıfı, İlk Giren İlk Çıkar (ing. FIFO – First Input First Output) veri yapısı (ing. Data Structure) işlevini sağlayan kuyruk sınıfıdır.

| C# | VB.NET |
| --- | --- |
| **public** **class** Queue :  ICollection,  IEnumerable,  ICloneable {  . . .  } | **Public** **Class** Queue  **Implements** ICollection,  IEnumerable,  ICloneable  . . .  **End** **Class** |

Kod 188, System.Collections.Queue Sınıf Tanımı

Aşağıda kuyruk veri yapısı örneklenmiştir.

Şekil 34, Kuyruk Veri Yapısı

Kuyruk veri yapısı, listeye eklenen son nesnenin liste sonuna alınması (Enqueue), listeden eleman çıkartmanın (Dequeue) sadece liste başından yapılabilmesini sağlamaktadır.

Yukarıda anlatılan işlevleri sağlayan Queue sınıfı üye bilgileri ve açıklamaları aşağıda gösterilmiştir.

| Üye Adı | Açıklama |
| --- | --- |
| Count | Kuyruk içerisinde bulunan eleman sayısını döndürür. |
| Dequeue | Liste başındaki elemanı çıkarır ve ilgili elemanı döndürür. |
| Enqueue | Liste sonuna eleman ekler. |
| Peek | Liste başındaki elemanı döndürür. |

Tablo 69, System.Collections.Queue Sınıfı Üye Bilgileri

Stack Kullanımı

Stack sınıfı, Son Giren İlk Çıkar (ing. LIFO – Last Input First Output) veri yapısı işlevini sağlayan yığın sınıfıdır.

| C# | VB.NET |
| --- | --- |
| **public** **class** Stack :  ICollection,  IEnumerable,  ICloneable {  . . .  } | **Public** **Class** Stack  **Implements** ICollection,  IEnumerable,  ICloneable  . . .  **End** **Class** |

Kod 189, System.Collections.Stack Sınıf Tanımı

Aşağıda yığın veri yapısı örneklenmiştir.

Şekil 35, Yığın Veri Yapısı

Yığın veri yapısı, listeye eleman ekleme işleminin (Push) liste başına, listeden eleman çıkarma (Pop) işleminin yine liste başından yapılmasını sağlamaktadır.

Yukarıda anlatılan işlevleri sağlayan Stack sınıfı üye bilgileri ve açıklamaları aşağıda gösterilmiştir.

| Üye Adı | Açıklama |
| --- | --- |
| Count | Yığın içerisinde bulunan eleman sayısını döndürür. |
| Pop | Liste başındaki elemanı çıkarır ve ilgili elemanı döndürür. |
| Push | Liste başına eleman ekler. |
| Peek | Liste başındaki elemanı döndürür. |

Tablo 70, System.Collections.Stack Sınıfı Üye Bilgileri

Jenerik Koleksiyonlar

System.Collections.Generic ve System.Collections.ObjectModel ad alanları içerisinde bulunan jenerik koleksiyon sınıfları, kitabın daha önceki bölümlerinde aktarılan jenerik tipler kavramı çerçevesinde hazırlanmış sınıflardır.

Jenerik olmayan koleksiyon sınıflarının neredeyse tamamı System.Object tipi ile çalışmaktadır. Koleksiyona eklenen elemanlar farklı tipte nesneler olsa dahi, eklenen tüm nesneler koleksiyon sınıfları tarafından polimorfizm kuralları ile yönetilmektedir. Benzer şekilde, koleksiyon içerisinde bulunan herhangi bir nesneye erişim tip dönüşümü / değişimi kuralları ile yönetilmektedir.

| C# | VB.NET |
| --- | --- |
| ArrayList list = **new** ArrayList();  list.Add(1);  list.Add(2);  list.Add(3);  // Tip değişimi yapılmaktadır.  **int** i = (**int**)list[0];  // Aşağıdaki ifade derleme hatasına // üretecektir.  **int** ik = list[0]; | **Dim** list **As** ArrayList  list = **New** ArrayList()  list.Add(1)  list.Add(2)  list.Add(3)  **Dim** i **As** **Integer**  ' Tip değişimi yapılmaktadır.  i = **CType**(list(0), **Integer**)  ' Aşağıdaki ifade derleme hatasına ' üretecektir.  **Dim** ik **As** **Integer** = list(0) |

Kod 190, Koleksiyon Tip Değişim Örneği

Örnek kod parçasında liste içerisine tam sayı değerleri eklenmiş olmasına rağmen, liste içerisindeki herhangi bir elemana erişildiğinde liste tarafından döndürülen Object nesnesidir.

Kitabın önceki bölümlerinde detayları ile aktarıldığı üzere, System.Int32 değer tip (ing. Value Type) olduğu için, örnekte yapılan işlem kutulama ve kutudan çıkarmadır. Çalışma zamanında yapılan bu işlemin performansa önemli ölçüde negatif etkisi bulunmaktadır.

Benzer şekilde, koleksiyon içerisinde bulunan elemanın istenilen tipte olmaması durumunda çalışma zamanında istisna fırlatılmaktadır.

| C# | VB.NET |
| --- | --- |
| ArrayList list = **new** ArrayList();  list.Add(1);  list.Add(2);  list.Add("3");  **int** i = (**int**)list[0];  // InvalidCast istisnası  // fırlatılacaktır.  **int** ik = (**int**)list[2]; | **Dim** list **As** ArrayList  list = **New** ArrayList()  list.Add(1)  list.Add(2)  list.Add("3")  **Dim** i **As** **Integer**  i = **CType**(list(0), **Integer**)  ' InvalidCast istisnası  ' fırlatılacaktır.  **Dim** ik **As** **Integer**  ik = **CType**(list(2), **Integer**) |

Kod 191, Tip Değişim İstisna Örneği

Yukarıdaki örnekte liste içerisindeki ilk iki eleman tam sayı tipinde olmasına rağmen üçüncü eleman karakter dizi tipindedir. Karakter dizi tipinde bir nesnenin tamsayı tipine dönüşümü otomatik olarak yapılamadığı için System.InvalidCastException istisnası CLR tarafından fırlatılmaktadır.

Örnekte, listeye sadece tamsayı tipinde olan nesnelerin eklenmesinin zorunlu kılınamayacağına dikkat edilmelidir.

Jenerik koleksiyonlar, koleksiyon içeriğini oluşturan elemanların tiplerinin derleme zamanında belirlenebilmesini sağlamaktadır.

| C# | VB.NET |
| --- | --- |
| // Jenerik liste sınıfı.  List<**int**> list = **new** List<**int**>();  list.Add(1);  list.Add(2);  // Aşağıdaki ifade derleme zamanı // hatası üretecektir. list isimli // nesne sadece tamsayı içeriğe // sahip olabilecek şekilde // oluşturulmuştır.  list.Add("3");  // Tip değişimi işlemine gerek  // kalmamıştır.  **int** i = list[0]; | ' Jenerik liste sınıfı.  **Dim** list **As** List(**Of** **Integer**)  list = **New** List(**Of** **Integer**)()  List.Add(1)  List.Add(2)  ' Aşağıdaki ifade derleme zamanı ' hatası üretecektir. list isimli ' nesne sadece tamsayı içeriğe ' sahip olabilecek şekilde ' oluşturulmuştır.  List.Add("3")  ' Tip değişimi işlemine gerek  ' kalmamıştır.  **Dim** i **As** **Integer** = list(0) |

Kod 192, Örnek Jenerik Koleksiyon Kullanımı

Jenerik Koleksiyon Bilgileri

Aşağıda jenerik olmayan koleksiyon tipleri ve her bir koleksiyon tipine karşılık gelen jenerik olan sürümü listelenmiştir.

| Tip | Jenerik Tip |
| --- | --- |
| ArrayList | List<T> |
| HashTable | Dictionary<T> |
| Queue | Queue<T> |
| SortedList | SortedList<T> |
| Stack | Stack<T> |
| StringCollection | List<string> |
| StringDictionary | Dictionary<string> |
| ListDictionary | Dictionary<T> |
| HybridDictionary | Dictionary<T> |
| OrderedDictionary | Dictionary<T> |
| NameValueCollection | Dictionary<T> |
| DictionaryEntry | NameValuePair<T> |

Tablo 71, Jenerik Olmayan ve Jenerik Koleksiyon Sınıf Bilgileri

System.Collections.ObjectModel ad alanı içerisinde bulunan jenerik koleksiyon sınıfları ve kullanım amaçları aşağıda listelenmiştir.

| Sınıf | Açıklama |
| --- | --- |
| Collection<T> | Yeni jenerik koleksiyon sınıf tanımları için temel işlevler içeren sınıftır. |
| KeyedCollection<T> | Anahtar bilgisi, eleman değeri içerisinde bulunan koleksiyonlar için temel işlevler içeren soyut sınıftır. |
| ReadOnlyCollection<T> | Jenerik salt okunur koleksiyonlar için temel işlevleri içeren sınıftır. |

Tablo 72, System.Collections.ObjectModel Ad Alanı Sınıf Bilgileri

System.Collections.ObjectModel ad alan içerisinde bulunan listelenen jenerik koleksiyon sınıflarının temel kullanım amacı, programcılar tarafından miras alınarak yeni jenerik koleksiyon sınıfları tanımlanmasıdır.

Örnek olması açısından aşağıda jenerik List sınıf tanımı gösterilmiştir.

C#

**public** **class** List<T> :

IList<T>, ICollection<T>, IEnumerable<T>, IList, ICollection, IEnumerable {

. . .

}

VB.NET

**Public** **Class** List(**Of** T)

**Implements** IList(**Of** T), ICollection(**Of** T), IEnumerable(**Of** T), IList, \_  
 ICollection, IEnumerable

. . .

**End** **Class**

Kod 193, Jenerik List Sınıf Tanımı

Sınıf tanımından da görüleceği üzere, jenerik List<T> sınıfı, gerek jenerik IList<T> arabirimi gerek jenerik olmayan IList arabirim gerçekleştirimini yapmaktadır.

Bütün jenerik koleksiyon sınıfları benzer şekilde jenerik ve jenerik olmayan koleksiyon arabirim gerçekleştirimini yapmaktadır. Böylece, örneğin, jenerik List sınıfı jenerik olmayan IList arabirimi ile de ifade edilebilmektedir.

| C# | VB.NET |
| --- | --- |
| List<**int**> list = **new** List<**int**>();  list.Add(1);  list.Add(2);  // List<T> sınıfı IList arabirim // gerçekleştirimini yaptığı için // atama işlemi herhangi bir tip // dönüşümüne gerek kalmadan // yapılabilmektedir.  IList jList = list;  **object** o = jList[0];  Console.WriteLine(o);  // 1 değeri yazılacaktır. | **Dim** list **As** List(**Of** **Integer**)  list = **New** List(**Of** **Integer**)()  List.Add(1)  List.Add(2)  ' List<T> sınıfı IList arabirim ' gerçekleştirimini yaptığı için ' atama işlemi herhangi bir tip ' dönüşümüne gerek kalmadan ' yapılabilmektedir.  **Dim** jList **As** IList = list  **Dim** o **As** **Object** = jList(0)  Console.WriteLine(o)  ' 1 değeri yazılacaktır. |

Kod 194, Jenerik Sınıflar ve Jenerik Olmayan Koleksiyon Arabirimleri

Jenerik sınıflar içerisinde KeyedCollection sınıfı pratikte kullanım kolaylığı sağlamaktadır. Aşağıdaki kod örneğinde bu sınıf kullanımı ile ilgili bazı açıklamalar bulunmaktadır.

| C# | VB.NET |
| --- | --- |
| // TC Kimlik No ve ad özelliklerini // içeren kişi sınıf tanımı // yapılmaktadır.  **class** Kisi {  **private** **long** tcKimlikNo;  **public** **long** TcKimlikNo {  **get** { **return** tcKimlikNo; }  **set** { tcKimlikNo = value;}  }  **private** **string** ad;  **public** **string** Ad {  **get** { **return** ad; }  **set** { ad = value; }  }  **public** Kisi(**long** tcKimlikNo,   **string** ad) {  **this**.ad = ad;  **this**.tcKimlikNo = tcKimlikNo;  }  } | ' TC Kimlik No ve ad özelliklerini ' içeren kişi sınıf tanımı ' yapılmaktadır.  **Class** Kisi  **Private** \_tcKimlikNo **As** **Long**  **Public** **Property** TcKimlikNo() **As** **Long**  **Get**  **Return** \_tcKimlikNo  **End** **Get**  **Set**(**ByVal** Value **As** **Long**)  \_tcKimlikNo = Value  **End** **Set**  **End** **Property**  **Private** \_ad **As** **String**  **Public** **Property** Ad() **As** **String**  **Get**  **Return** \_ad  **End** **Get**  **Set**(**ByVal** Value **As** **String**)  \_ad = Value  **End** **Set**  **End** **Property**  **Public** **Sub** **New**(**ByVal** tcKimlikNo   **As** **Long**,   **ByVal** ad **As** **String**)  **End** **Sub**  **End** **Class** |

C#

// KeyedCollection sınıfı miras alınarak KisiListesi isimli sınıf   
// tanımlanmıştır. long, anahtar tipini Kisi, liste içerisindeki eleman   
// tipini belirtmektedir.

**class** KisiListesi: KeyedCollection<**long**, Kisi> {

// Metot örtülerek liste için anahtar bilgisi belirtilmektedir.

**protected** **override** **long** GetKeyForItem(Kisi item) {

**return** item.TcKimlikNo;

}

}

KisiListesi list = = **new** KisiListesi();

Kisi k1 = **new** Kisi(59716271020, "Hasan");

Kisi k2 = **new** Kisi(56091234993, "Veli");

list.Add(k1);

list.Add(k2);

**string** ad;

ad = list[56091234993].Ad;

// ad, "Veli" değeri alacaktır.

// İlk elemanı listeden sil.

list.Remove(k1);

VB.NET

' KeyedCollection sınıfı miras alınarak KisiListesi isimli sınıf   
' tanımlanmıştır. long, anahtar tipini Kisi, liste içerisindeki eleman tipini

' belirtmektedir.

**Class** KisiListesi

**Inherits** KeyedCollection(**Of** **Long**, Kisi)

' Metot örtülerek liste için anahtar bilgisi belirtilmektedir.

**Protected** **Overrides** **Function** GetKeyForItem(**ByVal** item **As** Kisi) **As** **Long**

**Return** item.TcKimlikNo

**End** **Function**

**End** **Class**

**Dim** list **As** KisiListesi = **New** KisiListesi()

**Dim** k1 **As** Kisi = **New** Kisi(59716271020, "Hasan")

**Dim** k2 **As** Kisi = **New** Kisi(56091234993, "Veli")

list.Add(k1)

list.Add(k2)

**Dim** ad **As** **String** = list(56091234993).Ad

' ad, "Veli" değeri alacaktır.

' İlk elemanı listeden sil.

list.Remove(k1)

Kod 195, System.Collections.ObjectModel.KeyedCollection Kullanımı

Jenerik ve jenerik olmayan koleksiyon sınıfları programlama yaparken sık kullanılan sınıflardır. Kitabın bu bölümünde konu hakkında olabildiğince detaylı bilgi verilmeye çalışılmıştır.  
  
Koleksiyon sınıflarının ve arabirimlerinin pratikte bir diğer kullanımı da görsel kontrollere veri bağlanmasıdır. Gerek ASP.NET, gerek Windows Forms.NET kullanıcı ara yüz bileşenleri IEnumerable, IList, ICollection gibi koleksiyon sınıfları ile veri bağlamayı desteklemekte, bu yapıların kullanımı zaman ve emekten büyük ölçüde tasarruf edebilmeyi sağlamaktadır.  
  
Okuyucuların bir sonraki konuya geçmeden önce koleksiyonlar ile ilgili pratik çalışmalar yapması önerilmektedir.

İçinde yaşadığımız yeniçağda, zenginlik bilginin ürünüdür. Bilgi, ekonominin başlıca hammaddesi ve en önemli ürünü haline gelmiş bulunuyor. Günümüzde zenginlik yaratmak için gerek duyulan sermaye varlıkları arazi, bedensel emek, imalat ve fabrikalar değildir. Bunların yerini bilgi almış durumdadır.

Thomas A.Stewart

Bölüm

13

Girdi – Çıktı Yönetimi

S

ystem.IO ad alanı içerisinde bulunan ve girdi – çıktı (ing. Input – Output) yönetimini yapan sınıflar bu bölümün konusunu oluşturmaktadır.

GÇ (Girdi – Çıktı) yönetimi, .NET içerisinde farklı amaçlarla kullanılmaktadır. Özellikle GÇ işlemlerinin temelini oluşturan akımlar (ing. Stream) konusunun teorik düzeyde iyi anlaşılmış olması, kitabın ilerleyen bölümlerinde aktarılacak konular için önem taşımaktadır.

Genel Mimari

System.IO üst ad alanı, GÇ yönetiminin yapılmasını sağlayan tip tanımlarının bulunduğu ad alanıdır. System.IO altında bulunan alt ad alanları içerisinde, farklı amaçlar için kullanılabilecek tip tanımları bulunmaktadır.

| Ad Alanı | Açıklama |
| --- | --- |
| System.IO | Dosyalara yazma, okuma, akım nesneleri gibi GÇ yönetimi temel mimari yapısı ve sınıflarını içeren ad alanıdır. |
| System.IO.IsolatedStorage | Yalıtılmış depolama (ing. Isolated Storage) sınıf ve tip tanımlarının bulunduğu ad alanıdır. |
| System.IO.Compression | Veri sıkıştırma amacıyla kullanılabilecek sınıf ve tiplerin bulunduğu ad alanıdır. |
| System.IO.Ports | Seri port kontrolünün yapılabileceği sınıf ve tiplerin bulunduğu ad alanıdır. |

Tablo 73, System.IO ve Alt Ad Alan ve Açıklamaları

Programlama sırasında bir dosyanın okunması veya bir dosyaya bilgi yazılması en sık yapılan işlemlerindendir. Benzer şekilde, sistem hafızasına bilgi yazma ve hafızadan bilgi okuma, veri sıkıştırma gibi işlemlere de çoğu uygulama yazılımı tarafından ihtiyaç duyulmaktadır.

.NET tarafından sağlanan sınıflar ve sunulan GÇ mimarisi, uygulama yazılımları tarafından ihtiyaç duyulan GÇ işlevlerinin kolaylıkla yapılabilmesini sağlamaktadır.

Aşağıdaki tabloda .NET tarafından sağlanan temel GÇ sınıfları ve açıklamaları listelenmiştir.

| Sınıf | Açıklama |
| --- | --- |
| FileSystemInfo | Dosya veya klasörler hakkında bilgi almak için kullanılan temel sınıftır. FileInfo ve DirectoryInfo sınıfları bu sınıfı miras almaktadır. |
| FileInfo | Dosya hakkında bilgi almayı ve temel dosya işlemlerinin yapılmasını sağlayan özellik ve metotların bulunduğu sınıftır. |
| DirectoryInfo | Klasör hakkında bilgi almayı ve temel klasör işlemlerinin yapılmasını sağlayan özellik ve metotların bulunduğu sınıftır. |
| DriveInfo | Sürücü hakkında bilgi almayı sağlayan sınıftır. |
| File | FileInfo sınıfına benzer işlevleri sağlayan ve sadece statik metotlar içeren sınıftır. |
| Directory | DirectoryInfo sınıfına benzer işlevleri sağlayan ve sadece statik metotlar içeren sınıftır. |
| Path | Dosya veya klasör yolları ile ilgili işlemlerin yapılmasını kolaylaştıran işlevler sağlayan sınıftır. |
| FileSystemWatcher | Dosya sistemindeki değişiklikleri gözleyen sınıftır. |
| Stream | Soyut akım sınıfıdır. Tüm akım sınıfları bu sınıfı miras almaktadır. |
| MemoryStream | Hafıza akım sınıfıdır. |
| FileStream | Dosya akım sınıfıdır. |
| BufferedStream | Arabellek (ing. Buffer) desteği olan akım sınıfıdır. |
| TextReader TextWriter | Metin bazlı okuma / yazma işlevini sağlayan temel soyut sınıftır. |
| StreamReader StreamWriter | Metin bazlı okuma / yazma işlevini sağlayan temel sınıftır. Bu sınıf TextReader / TextWriter sınıfını miras almaktadır. |
| BinaryReader BinaryWriter | Temel tiplerin (ing. Primitve Types) ikil değerler olarak okunması / yazılması işlevini sağlayan sınıftır. |
| StringReader StringWriter | TextReader sınıfını miras alan ve karakter dizi değerlerden okuyabilen / karakter dizi değerlere yazabilen sınıftır. |
| IsolatedStorageFile | Yalıtılmış depolama alanını içerisinde dosya işlemleri yapabilmeyi sağlayan statik metotlar içeren sınıftır. |
| IsolatedStorageFileStream | Yalıtılmış depolama alanı içerisinde dosya işlemleri yapabilmeyi sağlayan akım sınıfıdır. |
| DeflateStream | Deflate veri sıkıştırma algoritması gerçekleştirimini yapan akım sınıfıdır. |
| GZipStream | GZip veri sıkıştırma algoritması gerçekleştirimini yapan akım sınıfıdır. |
| SerialPort | Bilgisayar seri port arabirimi üzerinde işlemleri yapabilmeyi sağlayan sınıftır. |

Tablo 74, Temel Girdi – Çıktı Sınıfları ve Açıklamaları

GÇ yönetimin yukarıdaki tabloda listelenen sınıflar yardımıyla ne şekilde yapılabileceği konusu hakkında detaylı bilgiler verilmeden önce karakter verilerin kodlaması ve kodlama standartları hakkında bilgiler sunulacaktır.

Veri Kodlama ve Standartları

GÇ yönetiminin temel işlevi veri ile çalışmaktır. Bilindiği gibi bilgisayar tarafından ele alınabilecek en küçük veri miktarı bit olarak ifade edilmektedir. Bir bit, 0 ve 1 değerlerini alabilmektedir.

0 ve 1 değerleri ile iki adet farklı durum ifade edilebilmektedir. Örneğin var, yok veya doğru, yanlış gibi. Bit sayısı ikiye çıktığında, iki bit ile ifade edilebilecek durum sayısı dört olmaktadır. Bit sayısının üç olması, sekiz farklı durumu ifade edebilmeyi sağlamaktadır. Genellemek gerekirse;

Durum Sayısı = 2 ^ Bit Sayısı (2 üzeri bit sayısı)

Formülü ile hesaplanabilir.

Veri Kodlama

Veri kodlama (ing. Encoding), verinin bir formattan bir başka formata dönüşümü olarak tanımlanmaktadır. Veri şifreleme, karakter kodlama, elektronik kodlama kavramları veri kodlamaya örnek olarak verilebilir.

Karakter Veri Kodlama

Karakter veri kodlama, sayısal değerlerin karakter değerlere çevrimidir.

Yukarıdaki konuda, bit kavramı ve ifade edilebilecek durumlar hakkında bilgiler sunulmuştu. Türkçe alfabesinde 29 harf bulunduğundan, bir başka anlatımla 29 farklı durumdan yola çıkılarak, en az 5 bit ile 29 durumun ifade edilebileceği görülmektedir.

Karakter veri kodlama standartlarının öncülerinden olan ASCII (American Standart Code for Information Interchange), ilk çalışma modelinde, bir karakterin 7bit ile ifade edilebileceğini belirtmektedir. Bu durumda, 2^7 = 128 (0-127) adet farklı karakterin 7bit uzunluğundaki veri ile ifade edilebilmesi sağlanmıştır. İngiliz alfabesinde bulunan 26 karaktere ek olarak kontrol karakterleri, farklı karakter şekilleri de 128 adet karaktere dahil edilmiştir.

Örneğin, karakter 'A' 65, 'B' 66, 'C' 67 sayıları ile ifade edilmiştir.

Bilindiği gibi bilgisayar için anlamlı en küçük veri miktarı 8bit içeren 1Byte'dır. ASCII standardında, 7bit kullanılmakta, kalan 1bit kullanılmamaktadır. Bu durumda, 1Byte ile ifade edilebilecek 256 karakterden sadece ilk bölüm (0–127) ASCII tarafından kullanılmaktadır.

İlk çalışma modelinde, 128 karakter içerisinde sadece İngilizce alfabesi karakter kümesi bulunmaktadır. ASCII tarafından belirlenen standart, farklı ülkelerin ve dillerin kullanmış olduğu karakterlere bir çözüm sağlayamadığı için, bilgisayar üreticileri kullanılmayan bölüme (128–255) farklı ülke ve dillere özgü karakterleri atamışlardır. Zaman içerisinde, belirli bir standardın olmamasından dolayı, aynı sayı değeri, örneğin 233, farklı ülkeler için farklı karakterleri ifade etmeye başlamıştır.

Bu problemi ortadan kaldırmak amacıyla ANSI (American National Standarts Institute) tarafından kod sayfaları (ing. Code Pages) belirleme çalışmaları başlamıştır.

Kod sayfalarının temel amacı, aynı veya benzer dillerin sayı – karakter eşlemelerini standart bir zemine oturtmayı sağlamaktır. Özellikle web sayfalarında, bazı karakterlerin kutucuklar şeklinde görüntülenmesi, hatalı kod sayfası kullanımına örnek olarak verilebilir.

Aşağıda örnek html (ing. Hypertext Markup Language) kaynak kodu başlık bilgileri gösterilmiştir.

Content-Type: text/html; charset=**iso-8859-9**

Content-Length: 5603

Sunucu tarafından Charset deyimi ile internet tarayıcısına iletilen bilgi, sayfa içeriğinin iso-8859-9 standartlarında kodlanmış olduğudur. Örneğin 8859–9 Türkçe karakter kümesini ifade ederken 8859–7 Yunan karakter kümesini ifade etmektedir.

Pratikte kod sayfaları kullanımı, doğru karakterlerin algılanması için yeterli olsa da, içerik ile birlikte kod sayfası bilgisinin de iletimi zorunludur.

ASCII ve ISO–8859 standartlarında olan eksiklerin giderilmesi amacıyla, Unicode olarak ifade edilen ve binlerce karakteri daha sağlıklı destekleyebilen bir standart belirlenmiş ve günümüzde kullanımı önerilmektedir.

Unicode standardı, geçmiş standartlara da destek verecek şekilde tasarlanmış olup aşağıda listelenen alt standartlardan oluşmaktadır.

* UTF–32  
  Bir karakterin 32bit ile ifade edildiği standarttır.
* UTF–16  
  Bir karakterin 16byte ile ifade edildiği standarttır.
* UTF–8  
  Bir karakterin 8, 16, 24, ve 48bit ile ifade edilebileceği standarttır. Örneğin ilk 8bit ASCII standartlarına karşılık gelmektedir.

.NET, UTF–16 standardını kullanmaktadır.

.NET Karakter Veri Kodlama Sınıfları

Önceki konularda teorik düzeyde aktarılan karakter kodlama standartları, .NET sınıf kütüphanesi içerisinde System.Text ad alanı içerisinde bulunan sınıflarla yönetilmektedir.

| Üye Adı | Açıklama |
| --- | --- |
| Encoding | Soyut karakter veri kodlama sınıfıdır. |
| ASCIIEncoding | ASCII standardını ifade eden karakter veri kodlama sınıfıdır. |
| UTF32Encoding | UTF–32 standardını ifade eden karakter veri kodlama sınıfıdır. |
| UnicodeEncoding | UTF–16 standardını ifade eden karakter veri kodlama sınıfıdır. |
| UTF8Encoding | UTF–8 standardını ifade eden karakter veri kodlama sınıfıdır. |

Tablo 75, Karakter Veri Kodlama Sınıfları ve Açıklamaları

Veri kodlama sınıfları direkt nesne oluşturarak kullanılabileceği gibi, Encoding sınıfı tarafından sağlanan statik özellik ve metotlar yardımıyla da kullanılabilir.

Encoding sınıfı temel üye bilgileri ve açıklamaları aşağıda gösterilmiştir.

| Üye Adı | Açıklama |
| --- | --- |
| ASCII | ASCIIEncoding nesnesi döndüren statik özelliktir. |
| UTF32 | UTF32Encoding nesnesi döndüren statik özelliktir. |
| Unicode | UnicodeEncoding nesnesi döndüren statik özelliktir. |
| UTF8 | UTF8Encoding nesnesi döndüren statik özelliktir. |
| GetEncodings | Tüm kodlamaları EncodingInfo dizisi olarak döndürür. |
| GetEncoding | Kod sayfa değeri veya kod sayfa adı ile Encoding nesnesi döndüren metottur. |
| GetString | Parametre olarak iletilen Byte dizi karşılığını karakter dizi değeri olarak döndüren metottur. |
| GetBytes | Parametre olarak iletilen karakter dizi karşılığını Byte dizi değeri olarak döndüren metottur. |
| GetByteCount | Parametre olarak iletilen karakter dizi değerinin kaç Byte ile ifade edildiğini döndürür. |

Tablo 76, System.Text.Encoding Sınıfı Üye Bilgileri

Aşağıda, Encoding sınıfı kullanımı ile ilgili çeşitli örnekler bulunmaktadır.

C#

Console.WriteLine("Karakter kodlama standartları ...");

// Encoding.GetEncodings metodu ile veri kodlama standartları hakkında bilgi   
// alınabilir.

**foreach** (EncodingInfo e **in** Encoding.GetEncodings()) {

Console.WriteLine("Ad: {0}",e.Name);

Console.WriteLine("Görünüm Ad: {0}", e.DisplayName);

Console.WriteLine("Kod Sayfa Değeri: {0}", e.CodePage);

}

Console.WriteLine("'Test' karakteri Byte Sayıları ...");

// 4 karakter uzunluğundaki bir karakter dizi değerin, farklı veri kodlama  
// standartlarında boyutu.

Console.WriteLine("ASCII: {0}", Encoding.ASCII.GetByteCount("Test"));

Console.WriteLine("UTF32: {0}", Encoding.UTF32.GetByteCount("Test"));

Console.WriteLine("UTF16: {0}", Encoding.Unicode.GetByteCount("Test"));

Console.WriteLine("UTF8: {0}", Encoding.UTF8.GetByteCount("Test"));

// Encoding.GetEncoding metodu ile farklı kod sayfalarına ait veri  
// kodlama standardı nesnesi oluşturulabilmektedir.

Encoding turkishEncoding = Encoding.GetEncoding("ISO-8859-9");

Console.WriteLine("Türkçe: {0}", turkishEncoding.GetByteCount("Test"));

// Türkçe ve UTF-8 kodlama standarlarında aynı karakter dizi değerin  
// byte dizi karşılığının alınması.

**byte**[] turkishBytes = turkishEncoding.GetBytes("Yazılım İşlevi");

**byte**[] utf8Bytes = Encoding.UTF8.GetBytes("Yazılım İşlevi");

Console.WriteLine("'Yazılım İşlevi' karakter sayıları ...");

Console.WriteLine("Türkçe Byte Sayısı: {0}", turkishBytes.Length);

Console.WriteLine("Utf8 Byte Sayısı: {0}", utf8Bytes.Length);

Console.WriteLine("Türkçe -> UTF8 Çevrimi ...");

Console.WriteLine(Encoding.UTF8.GetString(turkishBytes));

Console.WriteLine("UTF8 -> Türkçe Çevrimi ...");

Console.WriteLine(turkishEncoding.GetString(utf8Bytes));

VB.NET

Console.WriteLine("Karakter kodlama standartları ...")

' Encoding.GetEncodings metodu ile veri kodlama standartları hakkında bilgi

' alınabilir.

**For** **Each** e **As** EncodingInfo **In** Encoding.GetEncodings()

Console.WriteLine("Ad: {0}", e.Name)

Console.WriteLine("Görünüm Ad: {0}", e.DisplayName)

Console.WriteLine("Kod Sayfa Değeri: {0}", e.CodePage)

**Next**

Console.WriteLine("'Test' karakteri Byte Sayıları ...")

' 4 karakter uzunluğundaki bir karakter dizi değerin, farklı veri kodlama

' standartlarında boyutu.

Console.WriteLine("ASCII: {0}", Encoding.ASCII.GetByteCount("Test"))

Console.WriteLine("UTF32: {0}", Encoding.UTF32.GetByteCount("Test"))

Console.WriteLine("UTF16: {0}", Encoding.Unicode.GetByteCount("Test"))

Console.WriteLine("UTF8: {0}", Encoding.UTF8.GetByteCount("Test"))

' Encoding.GetEncoding metodu ile farklı kod sayfalarına ait veri

' kodlama standardı nesnesi oluşturulabilmektedir.

**Dim** turkishEncoding **As** Encoding = Encoding.GetEncoding("ISO-8859-9")

Console.WriteLine("Türkçe: {0}", turkishEncoding.GetByteCount("Test"))

' Türkçe ve UTF-8 kodlama standarlarında aynı karakter dizi değerin

' byte dizi karşılığının alınması.

**Dim** turkishBytes() **As** **Byte** = turkishEncoding.GetBytes("Yazılım İşlevi")

**Dim** utf8Bytes() **As** **Byte** = Encoding.UTF8.GetBytes("Yazılım İşlevi")

Console.WriteLine("'Yazılım İşlevi' karakter sayıları ...")

Console.WriteLine("Türkçe **Byte** Sayısı: {0}", turkishBytes.Length)

Console.WriteLine("Utf8 **Byte** Sayısı: {0}", utf8Bytes.Length)

Console.WriteLine("Türkçe -> UTF8 Çevrimi ...")

Console.WriteLine(Encoding.UTF8.GetString(turkishBytes))

Console.WriteLine("UTF8 -> Türkçe Çevrimi ...")

Console.WriteLine(turkishEncoding.GetString(utf8Bytes))

Kod 196, System.Text.Encoding Sınıfı Kullanım Örnekleri

Örnek kapsamında;

* Tüm veri kodlama standartları hakkında bilgi alma.
* Aynı karakter dizi değerinin farklı veri kodlama standartlarında boyutları.
* Türkçeye özgü karakterlerin farklı veri kodlama standartlarında ne şekilde ele alındığı.
* Aynı Byte dizi değerlerinin farklı veri kodlama standartlarındaki karakter karşılıkları.

Konularında bilgiler sunulmuştur.

Aşağıda kod parçası çıktısı gösterilmiştir.

Ekran Çıktısı 51, Karakter Veri Kodlama Örnek Çıktısı

Girdi – Çıktı İşlemleri ve Veri Kodlama

Aktarılan bilgiler ve sunulan kod parçası çıktısından da görüleceği üzere, bir karakter farklı veri kodlama standartlarında 1, 2 veya 4Byte uzunluğunda olabilmektedir.

Temel amacı veri ile çalışmak olan GÇ işlemlerinde, verinin karaktere dönüşümü sırasında kullanılan veri kodlama standardı önem taşımaktadır. Birçok GÇ sınıfı, yüklenmiş metotlarında parametre olarak veri kodlama nesnesini almaktadır.

Dosya Sistemi İşlemleri

Dosya sistemi hakkında bilgi almak, dosya kopyalama, silme, taşıma gibi temel dosya sistemi işlemlerinin yapılabilmesi amacıyla FileInfo ve DirectoryInfo sınıfları (1) ile File ve Directory sınıfları (2) System.IO ad alanı içerisinde bulunmaktadır.

Bu sınıfların (1) ve (2) sağladıkları işlevler neredeyse aynı olmakla birlikte, File ve Directory isimli sınıflar sadece statik özellik ve metotlar içermektedir.

System.IO ad alanı içerisinde bulunan ve konu ile ilgili sınıf hiyerarşisi aşağıda gösterilmiştir.

Şekil 36, System.IO Ad Alanı Dosya İşlemleri Sınıf Hiyerarşisi

Konu anlatımında, yukarıda (1) ve (2) ile ifade edilen sınıflardan sadece FileInfo ve DirectoryInfo sınıfları hakkında detaylı bilgi verilecek, gerektiği durumlarda benzer / farklı işlevlerin File ve Directory sınıfları ile nasıl sağlanacağı anlatılacaktır.

FileInfo ve DirectoryInfo sınıfları sırasıyla dosya ve klasörler ile ilgili işlevleri içermektedir. Her iki sınıfın da ata sınıfı olan soyut FileSystemInfo sınıfı temel üye bilgileri aşağıda listelenmiştir.

| Üye Adı | Açıklama |
| --- | --- |
| Attributes | Dosya veya klasör niteliklerini döndürür. (Gizli, salt okunur, vb.) |
| CreationTime | Dosya veya klasörün oluşturulma zamanını döndürür. |
| LastAccessTime | Dosya veya klasöre son erişim tarihini döndürür. |
| LastWriteTime | Dosya veya klasöre son yazma tarihini döndürür. |
| Exists | Dosya veya klasörün var olup olmadığını döndürür. |
| Extension | Dosya veya klasörün uzantısını döndürür. |
| FullName | Bütünleşik dosya / klasör adını döndürür. |
| Name | Dosya veya klasör adını döndürür. |

Tablo 77, System.IO.FileSystemInfo Sınıfı Üye Bilgileri

FileSystemInfo tarafından sağlanan işlevler ve bu işlevlerin farklı sınıflar ile nasıl kullanılabileceği konusunda aşağıdaki tabloda bilgiler sunulmuştur.

| İşlev | Yöntem |
| --- | --- |
| Dosya / klasör özelliklerinin alınması | File.GetAttributes / Directory.GetAttributes |
| Dosya / klasör hakkında zaman bilgilerinin kontrolü | File.GetCreationTime / Directory.GetCreationTime, File.GetLastAccessTime / Directory.GetLastAccessTime, … |
| Dosya / klasörün var olup olmadığının alınması | File.Exists / Directory.Exists |
| Dosya / klasör uzantısının alınması | Path.GetExtension |
| Dosya / klasör bütünleşik adının alınması | Path.GetFullPath |

Tablo 78, System.IO.FileSystemInfo ve Diğer Sınıf Metotları

Dosya İşlemleri

FileInfo sınıfı, dosya işlemlerini yapmayı sağlayan üyeler içeren GÇ sınıfıdır.

Aşağıda, FileInfo sınıfı üye bilgi ve açıklamaları listelenmiştir.

| Üye Adı | Açıklama |
| --- | --- |
| Directory | Dosyanın bulunduğu klasörün ifade edildiği DirectoryInfo nesnesidir. |
| DirectoryName | Dosyanın bulunduğu klasör adını döndürür. |
| Length | Dosya boyutunu döndürür. |
| IsReadOnly | Dosyanın salt okunur olup olmadığını döndürür. |
| AppendText | Dosyaya metin verisi ekleyebilmeyi sağlayan StreamWriter nesnesi döndürür. |
| CopyTo | Dosyanın istenilen hedefe kopyalanmasını sağlar. |
| Create | Dosyayı oluşturur ve FileStream nesnesi döndürür. |
| CreateText | Metin verisi yazmak amacıyla dosyayı oluşturur ve StreamWriter nesnesi döndürür. |
| Decrypt | Aktif kullanıcı tarafından şifrelenmiş dosyanın çözülmesini sağlar. |
| Encrypt | Dosyayı şifreler. |
| MoveTo | Dosyayı belirtilen hedefe taşır. |
| Open | Dosyayı açar ve FileStream nesnesi döndürür. |
| OpenRead | Dosyayı salt okunur olarak açar. |
| OpenText | Dosyayı metin verisini okumak amacıyla açar ve StreamReader nesnesi döndürür. |
| OpenWrite | Dosyayı sadece yazma amacıyla açar ve FileStream nesnesi döndürür. |
| Replace | Parametre olarak iletilen dosya üzerine mevcut dosyayı yazar. |

Tablo 79, System.IO.FileInfo Sınıfı Üye Bilgileri

C#

// FileInfo nesnesi oluştur.

FileInfo f = **new** FileInfo("c:\\test.txt");

// FileInfo.CreateText metodu kullanılarak dosyaya metin veri yazmak amacıyla   
// StreamWriter nesnesi oluşturulmaktadır.

**using** (StreamWriter sw = f.CreateText()) {

sw.WriteLine("Test 1");

sw.WriteLine("Test 2");

sw.Close();

}

// Dosyayı kopyala.

f.CopyTo("c:\\test.bak");

// Dosyayı sil.

f.Delete();

VB.NET

' FileInfo nesnesi oluştur.

**Dim** f **As** FileInfo = **New** FileInfo("c:\test.txt")

' FileInfo.CreateText metodu kullanılarak dosyaya metin veri yazmak amacıyla   
' StreamWriter nesnesi oluşturulmaktadır.

**Using** sw **As** StreamWriter = f.CreateText()

sw.WriteLine("Test 1")

sw.WriteLine("Test 2")

sw.Close()

**End** **Using**

' Dosyayı kopyala.

f.CopyTo("c:\test.bak")

' Dosyayı sil.

f.Delete()

Kod 197, System.IO.FileInfo Sınıfı Kullanım Örneği

FileInfo sınıfının bazı metotları akım nesneleri döndürmektedir. Akım nesneleri konusunda kitabın ilerleyen bölümlerinde detaylı bilgi sunulacaktır.

Klasör İşlemleri

DirectoryInfo sınıfı, klasör işlemlerini yapmayı sağlayan GÇ sınıfıdır.

Aşağıda, DirectoryInfo sınıfı üye bilgi ve açıklamaları listelenmiştir.

| Üye Adı | Açıklama |
| --- | --- |
| Parent | Üst klasörü DirectoryInfo nesnesi olarak döndürür. |
| Root | Üst klasörü karakter dizi değeri olarak döndürür. |
| Create | Klasör oluşturur. |
| CreateSubdirectory | Alt klasör oluşturur. |
| GetDirectories | Aktif klasör içerisindeki klasörleri döndürür. |
| GetFiles | Aktif klasör içerisindeki dosyaları döndürür. |
| GetFileSystemInfos | Aktif klasör içerisindeki dosya ve klasörleri FileSystemInfo nesneleri şeklinde döndürür. |
| MoveTo | Klasörü taşır. |

Tablo 80, System.IO.DirectoryInfo Sınıfı Üye Bilgileri

Aşağıdaki kod parçasında, parametre olarak iletilen klasör içerisinde bulunan tüm dosyaları döndüren metot gösterilmiştir. Metodun, alt klasörleri de aradığına dikkat edilmelidir.

C#

**static** FileInfo[] TumDosyalariAl(**string** ustKlasor) {

DirectoryInfo dirInfo = **new** DirectoryInfo(ustKlasor);

// Alt klasörleri al.

DirectoryInfo[] dirList = dirInfo.GetDirectories();

List<FileInfo> list = **new** List<FileInfo>();

// Klasör içerisindeki dosyaları al.

FileInfo [] fileList = dirInfo.GetFiles();

list.AddRange(fileList);

**foreach** (DirectoryInfo info **in** dirList)

list.AddRange(TumDosyalariAl(info.FullName));

**return** list.ToArray();

}

VB.NET

**Shared** **Function** TumDosyalariAl(**ByVal** ustKlasor **As** **String**) **As** FileInfo()

**Dim** dirInfo **As** DirectoryInfo = **New** DirectoryInfo(ustKlasor)

' Alt klasörleri al.

**Dim** dirList() **As** DirectoryInfo = dirInfo.GetDirectories()

**Dim** list **As** List(**Of** FileInfo) = **New** List(**Of** FileInfo)()

' Klasör içerisindeki dosyaları al.

**Dim** fileList() **As** FileInfo = dirInfo.GetFiles()

List.AddRange(fileList)

**For** **Each** info **As** DirectoryInfo **In** dirList

List.AddRange(TumDosyalariAl(info.FullName))

**Next**

**Return** List.ToArray()

**End** **Function**

Kod 198, System.IO.DirectoryInfo Kullanım Örneği

Listelenen kaynak kodlar \Bölüm 13\DirectoryInfo klasöründe bulunmaktadır.

Sürücü İşlemleri

DriveInfo sınıfı, sürücü / sürücüler hakkında bilgi almayı sağlayan üyelere sahiptir.

| Üye Adı | Açıklama |
| --- | --- |
| AvailableFreeSpace | Kullanılabilir disk alanını döndürür. |
| DriveFormat | Sürücü disk yapısını döndürür. (Örneğin FAT32, NTFS, …) |
| DriveType | Sürücü tipini döndürür. (Örneğin CD-ROM, Sabit, …) |
| Name | Sürücü adını döndürür. |
| TotalFreeSpace | Toplam boş disk alanı döndürür. |
| TotalSize | Toplam disk alanını döndürür. |
| VolumeLabel | Disk etiket bilgisini döndürür. |
| IsReady | Sürücünün hazır olup olmadığını döndürür. |
| GetDrives | Sistemdeki tüm sürücüleri döndüren statik metottur. |

Tablo 81, System.IO.DriveInfo Sınıfı Üye Bilgileri

| C# | VB.NET |
| --- | --- |
| DriveInfo [] list;  // Sistemde bulunan sürücüleri al.  list = DriveInfo.GetDrives();  **foreach** (DriveInfo i **in** list) {  Console.WriteLine(i.Name);  Console.WriteLine(i.IsReady);  **if** (i.IsReady)  Console.WriteLine(i.DriveFormat);  **else**  Console.WriteLine("Hazır değil");  Console.WriteLine(i.DriveType);  } | **Dim** list() **As** DriveInfo  ' Sistemde bulunan sürücüleri al.  list = DriveInfo.GetDrives()  **For** **Each** i **As** DriveInfo **In** list  Console.WriteLine(i.Name)  Console.WriteLine(i.IsReady)  **If** i.IsReady **Then**  Console.WriteLine(i.DriveFormat)  **Else**  Console.WriteLine("Hazır değil")  **End** **If**  Console.WriteLine(i.DriveType)  **Next** |

Kod 199, System.IO.DriveInfo Sınıfı Kullanım Örneği

Path Sınıfı ve Kullanım Örnekleri

System.IO.Path sınıfı, dosya veya klasör isimleri üzerinde işlemler yapabilmeyi kolaylaştıran statik üyeler ve sabitler içermektedir.

Aşağıda Path sınıfı temel üye bilgi ve açıklamaları listelenmiştir.

Türkçe karşılığı ile yol olarak ifade edilebilecek Path, dosya veya klasör ismini belirten kavram olarak algılanmalıdır.  
  
**Not:** Aşağıda parantez içerisinde bulunan karakterler, Windows işletim sistemi için döndürülen değerleri ifade etmektedir.

| Üye Adı | Açıklama |
| --- | --- |
| AltDirectorySeparatorChar | Alternatif klasör ayıraç karakterini döndürür. ( / ) |
| DirectorySeparatorChar | Klasör ayıraç karakterini döndürür. ( \ ) |
| PathSeparator | Ortam değişkenlerinde (ing. Environment Variables) farklı yol değerlerini ayıran karakteri döndürür. ( ; ) |
| VolumeSeparatorChar | Sürücü ayıraç karakterini döndürür. ( : ) |
| ChangeExtension | Parametre olarak iletilen yol ve uzantı bilgisi bazında, uzantısı değiştirilmiş yol değerini döndürür. |
| Combine | Parametre olarak iletilen iki yol değerini birleştirir. |
| GetDirectoryName | Parametre olarak iletilen yol değerinden, klasör bilgisini döndürür. |
| GetExtension | Parametre olarak iletilen yol değerinden, uzantı bilgisini döndürür. |
| GetFileName | Parametre olarak iletilen yol değerinden, dosya adı bilgisini döndürür. |
| GetFileNameWithoutExtension | Parametre olarak iletilen yol değerinden, uzantısı olmadan dosya adını döndürür. |
| GetInvalidFileNameChars | Dosya isimlerinde kullanılamayacak olan karakterleri döndürür. |
| GetInvalidPathChars | Yol isimlerinde kullanılamayacak olan karakterleri döndürür. |
| GetPathRoot | Parametre olarak iletilen yol değerinden, eğer varsa, kök klasör değerini döndürür. |
| GetRandomFileName | Rasgele üretilmiş dosya adı döndürür. |
| GetTempFile | Disk üzerinde tekil isim ile geçici dosya oluşturur ve oluşturulan dosya ismini döndürür. |
| GetTempPath | Sistemde tanımlı olan geçici klasör bilgisini döndürür. |
| HasExtension | Parametre olarak iletilen yol değerinin uzantı içerip içermediğini döndürür. |

Tablo 82, System.IO.Path Sınıfı Üye Bilgileri

System.IO.Path sınıfının sağladığı en önemli işlevsellik, farklı platformlar için, platforma özgü olarak çalışmasıdır.   
  
Örneğin Windows işletim sistemi için \ karakteri ayıraç olarak kullanılmasına rağmen, Unix tabanlı işletim sistemlerinde yol ayıracı / karakteridir. Path.DirectorySeparatorChar özelliği, çalışılan platform bazında doğru karakteri döndürecektir.

Özel Klasörler

Özel klasörler, yerleşimleri Windows işletim sistemi kurulum ve konfigürasyonu veya giriş yapmış aktif kullanıcı bazında farklılık gösterebilen klasörlerdir.

Örneğin özel bir klasör olan "Dokümanlarım" klasörü, sisteme giriş yapmış kullanıcının dokümanlarını saklaması için önerilen yerleşimi ifade etmektedir. Benzer şekilde, "Program Dosyaları" klasörü, programların yüklenilmesinin önerildiği yerleşimi göstermektedir.

System.Environment.SpecialFolder sayılabilir tipi özel klasörleri belirtmek için kullanılmaktadır.

| C# | VB.NET |
| --- | --- |
| **public** **enum** SpecialFolder {  ApplicationData = 0x1a,  CommonApplicationData = 0x23,  CommonProgramFiles = 0x2b,  Cookies = 0x21,  Desktop = 0,  DesktopDirectory = 0x10,  Favorites = 6,  . . .  } | **Public** **Enum** SpecialFolder  ApplicationData = &H1A  CommonApplicationData = &H23  CommonProgramFiles = &H2B  Cookies = &H21  Desktop = 0  DesktopDirectory = &H10  Favorites = 6  . . .  **End** **Enum** |

Kod 200, System.Environment.SpecialFolder Tip Tanımı

System.Environment.GetFolderPath metodu, özel klasör yerleşimlerini döndürmek için kullanılabilir.

Aşağıdaki örnekte "Dokümanlarım" özel klasör yerleşimi alınmaktadır.

C#

Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments);

// ör. C:\Documents and Settings\Kullanıcı\My Documents

VB.NET

Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments)

' ör. C:\Documents and Settings\Kullanıcı\My Documents

Kod 201, Özel Klasör Yerleşiminin Alınması

Dosya Sisteminin İzlenmesi

System.IO.FileSystemWatcher sınıfı, belirtilen klasör içerisinde olan değişikliklerin izlenmesi amacıyla kullanılabilir.

Değişikliklere örnek olarak, izlenen klasör içerisinde bulunan dosyaların değişmesi, yeni dosya oluşturulması veya var olan dosya / klasörün yeniden adlandırılması verilebilir.

FileSystemWatcher sınıfı, örneğin, uygulama konfigürasyon dosyalarında olan değişiklikleri izlemek için kullanılabilir. Uygulama, performansı artırmak amacıyla, açılışta konfigürasyon bilgilerini hafızaya alabilir. Konfigürasyon dosyasında olabilecek değişiklikler FileSystemWatcher sınıfı ile izlenebilir ve konfigürasyon bilgileri gerektiğinde dosyadan yeniden yüklenebilir.

FileSystemWatcher sınıfı üye bilgi ve açıklamaları aşağıda gösterilmiştir.

| Üye Adı | Açıklama |
| --- | --- |
| EnableRaisingEvents | True değerinin atanması durumunda izleme işlemi başlar ve değişiklik durumunda ilgili olaylar tetiklenir. |
| Filter | İzlenmesi istenen dosyaların uzantısını belirtmek için kullanılmalıdır. Ön tanımlı değeri \*.\* tüm dosyaları belirtmektedir. |
| IncludeSubdirectories | Alt klasörlerin izlenip izlenmeyeceğini belirtmek için kullanılmalıdır. |
| NotifyFilter | Hangi durumlarda (örneğin sadece yeniden adlandırma gibi) değişiklik olaylarının tetiklenmesi gerektiğini belirtir. |
| Path | İzleme yapılması gereken klasörü belirtir. |
| WaitForChanged | İzleme işlemini senkron olarak başlatır. |

Tablo 83, System.IO.FileSystemWatcher Sınıfı Üye Bilgileri

FileSystemWatcher kullanımını örnekleyen kod parçası aşağıda gösterilmiştir.

C#

// Dosya yeniden adlandırma işlemi.

**static** **void** w\_Renamed(**object** sender, RenamedEventArgs e) {

Console.WriteLine("Yeniden adlandırma: {0}->{1}", e.OldName, e.Name);

}

// Herhangi dosya işlemi.

**static** **void** DosyaIslemi(**object** sender, FileSystemEventArgs e) {

Console.WriteLine("Dosya işlemi bilgileri:");

Console.WriteLine("İşlem: {0}", e.ChangeType);

Console.WriteLine("Bütünleşik dosya adı: {0}", e.FullPath);

Console.WriteLine("Dosya adı: {0}", e.Name);

}

// Hata durumunda çalışacak metot.

**static** **void** w\_Error(**object** sender, ErrorEventArgs e) {

Console.WriteLine("Hata: {0}", e.GetException().Message);

}

**static** **void** Main(**string**[] args) {

// FileSystemWatcher nesnesi c:\test isimli klasör içerisindeki  
 // uzantısı .config olan dosyaları izleyecek şekilde oluşturulmaktadır.

FileSystemWatcher w;

w = **new** FileSystemWatcher(@"c:\test", "\*.config");

// İstenilen olaylara metot atamaları yapılmaktadır.

w.Error += w\_Error;

w.Created += DosyaIslemi;

w.Deleted += DosyaIslemi;

w.Changed += DosyaIslemi;

w.Renamed += w\_Renamed;

// EnableRaisingEvent özelliğine True değerinin atanması, izleme işlemini  
 // başlatmaktadır.

w.EnableRaisingEvents = **true**;

Console.WriteLine("İzlenen klasör: {0}", w.Path);

Console.WriteLine("İzlemeyi durdurmak için <enter> tuşu ...");

Console.ReadLine();

// İzleme işlemi uygulama kapanmadan önce sonlandırılmaktadır.

w.EnableRaisingEvents = **false**;

}

VB.NET

' Dosya yeniden adlandırma işlemi.

**Shared** **Sub** w\_Rename(**ByVal** sender **As** **Object**, **ByVal** e **As** RenamedEventArgs)

Console.WriteLine("Yeniden adlandırma: {0}->{1}", e.OldName, e.Name)

**End** **Sub**

' Herhangi dosya işlemi.

**Shared** **Sub** DosyaIslemi(**ByVal** sender **As** **Object**,   
 **ByVal** e **As** FileSystemEventArgs)

Console.WriteLine("Dosya işlemi bilgileri:")

Console.WriteLine("İşlem: {0}", e.ChangeType)

Console.WriteLine("Bütünleşik dosya adı: {0}", e.FullPath)

Console.WriteLine("Dosya adı: {0}", e.Name)

**End** **Sub**

' Hata durumunda çalışacak metot.

**Shared** **Sub** w\_Error(**ByVal** sender **As** **Object**, **ByVal** e **As** ErrorEventArgs)

Console.WriteLine("Hata: {0}", e.GetException().Message)

**End** **Sub**

**Shared** **Sub** Main(**ByVal** args() **As** **String**)

' FileSystemWatcher nesnesi c:\test isimli klasör içerisindeki  
 ' uzantısı .config olan dosyaları izleyecek şekilde oluşturulmaktadır.

**Dim** w **As** FileSystemWatcher

w = **New** FileSystemWatcher("c:\test", "\*.config")

' İstenilen olaylara metot atamaları yapılmaktadır.

**AddHandler** w.Error, **AddressOf** w\_Error

**AddHandler** w.Created, **AddressOf** DosyaIslemi

**AddHandler** w.Deleted, **AddressOf** DosyaIslemi

**AddHandler** w.Changed, **AddressOf** DosyaIslemi

**AddHandler** w.Renamed, **AddressOf** w\_Rename

' EnableRaisingEvent özelliğine True değerinin atanması, izleme işlemini  
 ' başlatmaktadır.

w.EnableRaisingEvents = **True**

Console.WriteLine("İzlenen klasör: {0}", w.Path)

Console.WriteLine("İzlemeyi durdurmak için <enter> tuşu ...")

Console.ReadLine()

' İzleme işlemi uygulama kapanmadan önce sonlandırılmaktadır.

w.EnableRaisingEvents = **False**

**End** **Sub**

Kod 202, System.IO.FileSystemWatcher Sınıfı Kullanım Örneği

Listelenen kaynak kodlar \Bölüm 13\FileSystemWatcher klasöründe bulunmaktadır.

Akımlar

Akımlar (ing. Streams) ile hedeflenen, verinin okunması veya yazılması işleminin akım üzerinden standardize edilmiş yöntemler kullanarak yapılabilmesinin sağlanmasıdır.

Verinin farklı şekillerde depolanmasını sağlayabilen farklı akımlar bulunmaktadır. Örneğin dosya akımları, verinin dosyaya yazılması / okunması işlevini sağlamaktadır. Benzer şekilde hafıza akımları, verinin hafızaya yazılması / okunması amacıyla kullanılabilir.

Akım kavramı ile sadece verinin depolanması anlaşılmamalıdır. Akımlar, veri iletimi amacıyla da kullanılabilir. Örneğin USB port üzerinden veri alışverişi yapmak amacıyla rahatlıkla .NET akım mimarisi kullanılabilir.

.NET Akım Mimarisi

.NET akım mimarisi içerisinde bulunan System.IO.Stream sınıfı, temel soyut akım sınıfıdır. Stream sınıfı tarafından tanımlanan bir çok soyut üye, farklı akım sınıfları tarafından örtülerek, akımın amacına uygun olarak veri işlemlerinin yapılmasını sağlamaktadır.

Aşağıda .NET akım mimarisi ve sınıf hiyerarşisi gösterilmiştir.

Şekil 37, .NET Akım Sınıfları

Stream sınıfı, IDisposable arabirim gerçekleştirimini yapmaktadır. Böylece, yönetilmeyen kaynakların akım tarafından kullanılması durumunda gerekli temizleme yapılabilmektedir.

| C# | VB.NET |
| --- | --- |
| **public** **abstract** **class** Stream :  MarshalByRefObject,  IDisposable {  . . .  } | **Public** **MustInherit** **Class** Stream  **Inherits** MarshalByRefObject  **Implements** IDisposable  . . .  **End** **Class** |

Kod 203, System.IO.Stream Sınıfı Tanımı

Akım nesnelerinden veri okuma ve akım nesnelerine veri yazma, akım üyeleri kullanılarak yapılabileceği gibi, aynı işlev akım okuyucu ve yazıcı sınıfları kullanılarak da sağlanabilir.

Akım okuyucu ve yazıcı sınıfları, .NET akım mimarisi içerisinde tanımlı olan ve akımlardan veri okuma / yazma işlemini daha standardize yöntemler ile yapabilmeyi sağlayan sınıflardır.

Akım okuyucu ve yazıcı sınıflar ve sınıf hiyerarşisi aşağıda gösterilmiştir.

Şekil 38, Akım Okuyucu - Yazıcı Sınıfları

System.IO.TextReader ve System.IO.TextWriter sınıfları, sırasıyla temel soyut akım okuyucu ve yazıcı sınıflarıdır. BinaryReader ve BinaryWriter sınıflarının kullanım amacı, temel tiplerin (tam sayı, sayılabilir, vb.) akımlardan okunabilmesi ve akımlara ikil olarak yazılabilmesidir.

.NET akım mimarisinin olabildiğince tüm detayları ile öğrenilmesi ve kavranması, farklı programlama konularının daha iyi anlaşılabilmesi için büyük önem taşımaktadır. Örneğin nesne serilizasyonu veya ağ üzerinden veri iletimi gibi işlemlerde farklı akım sınıfları kullanılmaktadır. Benzer şekilde, System.Security ad alanı içerisinde tanımlanmış olan ve Stream sınıfını miras alan CryptoStream sınıfı, veri şifreleme içi kullanılmaktadır.  
  
Konunun okuyucuya daha iyi aktarımının sağlanabilmesi amacıyla, öncelikli olarak temel akım sınıfı ile akım okuyucu / yazıcı sınıfları hakkında teorik düzeyde bilgiler sunulacaktır. Daha sonra System.IO ad alanı içerisinde tanımlı farklı akım sınıfları ve kullanım detayları hakkında bilgi ve örnekler aktarılacaktır.

Akımlar Temel Özellik ve İşlemleri

Akım nesneleri aşağıdaki işlevler ve kavramları içermektedir.

* Akım pozisyonu; Okuma ve yazma işleminin başlayacağı konumu belirtir değerdir. Genellikle, akım pozisyonu sıfır ile akım veri uzunluğu değeri aralığında olmaktadır.   
    
  Akım pozisyonu, akımdan veri okuma ve yazma işlemlerinde otomatik olarak ilerletilmektedir.
* Akım nesnesinden okuma; Akım nesnesi verisini istenilen miktarda okumayı ifade etmektedir.
* Konumlanabilme; Akım pozisyonunu istenilen geçerli bir değere değiştirebilmeyi ifade etmektedir.
* Akım boyutu; Akımın sahip olduğu verinin byte cinsinden uzunluğunu ifade etmektedir.
* Akım nesnesine yazma; Akım içerisine istenilen miktarda veriyi eklemeyi ifade etmektedir.

Yukarıda listelenen kavramların tamamı, Stream sınıfı tarafından soyut üyeler şeklinde ifade edilmektedir. Farklı akım sınıfları, Stream sınıfı tarafından tanımlanan üyeleri örterek farklı davranışlar sergileyebilmektedir. Örneğin dosya akımları için konumlanabilme işlevi desteklenmesine rağmen, ağ üzerinden veri iletimi veya veri şifreleme amacıyla kullanılan akım sınıflarında bu işlev desteklenmeyebilmektedir.

| Üye Adı | Açıklama |
| --- | --- |
| CanRead | Akımdan veri okunup okunamayacağını döndürür soyut özelliktir. |
| CanSeek | Akım sınıfının, veri üzerinde farklı pozisyona konumlanabilmeyi destekleyip desteklemediğini döndürür soyut özelliktir. |
| CanTimeout | Akım sınıfının zaman aşımını destekleyip desteklemediğini döndürür soyut özelliktir. |
| CanWrite | Akıma veri yazılıp yazılamayacağını döndürür soyut özelliktir. |
| Length | Akım boyutunu byte cinsinden döndüren soyut özelliktir. |
| Position | Akım pozisyonunu döndüren soyut özelliktir. |
| Close | Akımı kapatır ve akım ile ilgili kaynakların serbest bırakılmasını sağlar. |
| Flush | Ön bellekte tutulan verinin akım tipine göre gerçek kaynağa yazılmasını sağlayan soyut metottur. |
| Read | Akımdan veri okumayı sağlayan soyut metottur. |
| ReadByte | Akımdan bir byte uzunluğunda veri okumayı sağlayan soyut metottur. Akımda okunacak veri yok ise -1 değeri döndürülecektir. |
| Seek | Veri üzerinde istenilen pozisyona konumlanabilmeyi sağlayan soyut metottur. |
| SetLength | Akım boyutunu değiştirmeyi sağlayan soyut metottur. |
| Write | Akıma veri yazmayı sağlayan soyut metottur. |
| WriteByte | Akıma 1 byte uzunluğunda veri yazmayı sağlayan soyut metottur. |

Tablo 84, System.IO.Stream Sınıfı Üye Bilgileri

Akım Nesneleri Oluşturma ve Kullanım Önerileri

Soyut bir sınıf olan Stream sınıfı IDisposable arabirim gerçekleştirimini yapmaktadır. Bilindiği üzere IDisposable arabiriminin kullanım amacı, CLR tarafından yönetilmeyen (ing. Unmanaged) kaynakların nesne yok edilmeden önce serbest bırakılabilmesinin sağlanmasıdır.

Örneğin bilgisayar dosya sistemi işletim sistemi tarafından yönetilmektedir. Dosyalardan veri okuma / dosyalara veri yazma amacıyla kullanılan FileStream akım nesnesi, yok edilmeden önce işletim sistemi ile haberleşmeli ve işletim sistemine gerekli komutları iletebilmelidir.

Stream.Close, amacı ve içeriği farklı akım sınıflarında değişebilmekle birlikte, yönetilmeyen kaynakların serbest bırakıldığı metottur. IDisposable.Dispose metodu içerisinden de bu metot çağrılmaktadır. Akımı kapatan metot olarak tanımlanabilecek bu metodun, akım nesnesi kullanımı tamamlandıktan sonra, nesne yok edilme süreci beklenmeden, programcı tarafından çağrılması önerilmektedir.

Aşağıda, akım nesnelerinin genel kullanımı için önerilen yöntem gösterilmiştir.

C#

**using** (MemoryStream s = **new** MemoryStream()) {

// Akım nesnesi işlemleri burada yapılabilir.  
 // using bloğundan çıkmadan önce Close metodunun çağrılması   
 // önerilmektedir.

s.Close();

}

VB.NET

**Using** s **As** MemoryStream = **New** MemoryStream()

' Akım nesnesi işlemleri burada yapılabilir.  
 ' Using bloğundan çıkmadan önce Close metodunun çağrılması   
 ' önerilmektedir.

s.Close()

**End** **Using**

Kod 204, Akım Nesnesi Oluşturma, Kullanma ve Kapatma Yöntemi

Akım kullanımın using bloğu içerisinde olması ve bloktan çıkılmadan hemen önce Close metodunun çağrılması, blok içerisinde bir istisna fırlatılması durumunda IDisposable aracılığı ile, istisna fırlatılmaması durumunda Close metodu ile gerekli kaynakların akım nesnesi tarafından serbest bırakılabilmesini en etkin şekilde garanti altına almaktadır.

Okuma ve Yazma

Akımdan veri okuma ve akıma veri yazma işlemi sırasıyla Read ve Write metotları ile yapılmaktadır.

Aşağıda bu metotların tanımları ve parametre kullanım bilgileri gösterilmiştir.

C#

// buffer içerisine okuma yapılmaktadır. offset parametresi, buffer dizisi  
// içerisine yapılan okumanın dizinin hangi pozisyonundan başlayacağını,  
// count parametresi en fazla kaç byte okunması gerektiğini belirtmektedir.  
// İşlem sonucunda buffer dizisi içeriği, offset değerinden başlayarak   
// count değeri kadar akım verisi ile değiştirilecektir.  
  
// Read metodu, gerçekte okunan veri miktarını byte cinsinden döndürmektedir.

**public** **abstract** **int** Read(**byte**[] buffer, **int** offset, **int** count);

// buffer içerisinde bulunan veri akıma yazılmaktadır. Yazma işlemi offset   
// ile belirtilen değerden count sayısı kadar byte'ın buffer dizisinden akıma  
// yazılması şeklinde olacaktır.

**public** **abstract** **void** Write(**byte**[] buffer, **int** offset, **int** count);

VB.NET

' buffer içerisine okuma yapılmaktadır. offset parametresi, buffer dizisi  
' içerisine yapılan okumanın dizinin hangi pozisyonundan başlayacağını,  
' count parametresi kaç byte okunması gerektiğini belirtmektedir.  
' İşlem sonucunda buffer dizisi içeriği, offset değerinden başlayarak   
' count değeri kadar akım verisi ile değiştirilecektir.

**Public** **MustOverride** **Function** Read(**ByVal** buffer **As** **Byte**(), \_  
 **ByVal** offset **As** **Integer**, \_  
 **ByVal** count **As** **Integer**) **As** **Integer**

' buffer içerisinde bulunan veri akıma yazılmaktadır. Yazma işlemi offset   
' ile belirtilen değerden count sayısı kadar byte'ın buffer dizisinden akıma  
' yazılması şeklinde olacaktır.

**Public** **MustOverride** **Sub** Write(**ByVal** buffer **As** **Byte**(), \_  
 **ByVal** offset **As** **Integer**, \_  
 **ByVal** count **As** **Integer**)

Kod 205, Stream.Read ve Stream.Write Metot Tanımları

Aşağıda, akım nesnesinden okuyan ve okuduğu akım içeriğini konsol penceresinde gösteren kod parçası bulunmaktadır.

C#

**static** **void** StreamOku(Stream s, **int** boyut) {

**byte**[] b = **new** **byte**[boyut];

// İlk pozisyona konumlan.

s.Position = 0;

// Akımdan, b isimli byte dizi içerisine 0 pozisyonundan başlayarak  
 // en fazla b.Length kadar veriyi oku. Read metodu, gerçekte okunan veriyi  
 // döndürmektedir.

**int** toplam;

**while** ((toplam = s.Read(b, 0, b.Length)) > 0) {

// Console.WriteLine ( ... )

}

}

VB.NET

**Shared** **Sub** StreamOku(**ByVal** s **As** Stream, **ByVal** boyut **As** **Integer**)

**Dim** b(boyut - 1) **As** **Byte**

**'** İlk pozisyona konumlan.

s.Position = 0

' Akımdan, b isimli byte dizi içerisine 0 pozisyonundan başlayarak  
 ' en fazla b.Length kadar veriyi oku. Read metodu, gerçekte okunan veriyi  
 ' döndürmektedir.

**Dim** toplam **As** **Integer** = s.Read(b, 0, b.Length)

**While** toplam > 0

' Console.WriteLine ( ... )

toplam = s.Read(b, 0, b.Length)

**End** **While**

**End** **Sub**

Kod 206, Örnek Stream.Read İşlemi

Akım nesneleri ile veri okuma ve yazma işleminde dikkat edilmesi gereken en önemli konulardan birisi de, metin veri ile çalışılırken kullanılan karakter veri kodlama standardıdır. Kitabın daha önceki konularında da detayları ile aktarıldığı üzere, seçilen karakter veri kodlama standardı sayı – karakter eşliği ve bir karakterin kaç byte uzunluğunda olduğunu değiştirmektedir.  
  
Akım sınıflarının çoğunluğu, nesne oluşturma sırasında karakter veri kodlama standardının (ing. Encoding nesnesi) yapıcı metoda parametre olarak iletimini desteklemektedir.

Akımlar, farklı işlevleri sağlamak amacıyla yapılan tüm GÇ işlemlerinde en temel yapılardır. Aşağıdaki örnekte, belirtilen web sayfasına POST işleminin yapılması ve sonucun konsol penceresine iletimi gösterilmektedir.

C#

**using** System.Net;

// İstek nesnesini oluştur.

HttpWebRequest istek;

istek = HttpWebRequest.Create("http://www.dotnetturk.com") **as** HttpWebRequest;

istek.Method = "POST";

// Girdi akım nesnesi alınmakta ve Write metodu ile istenilen bilgi   
// akım nesnesine yazılmaktadır.

Stream girdi = istek.GetRequestStream();

**byte** [] girdiBytes = Encoding.UTF8.GetBytes("Ara=deneme&Tip=1");

girdi.Write(girdiBytes, 0, girdiBytes.Length);

girdi.Close();

WebResponse resp = istek.GetResponse();

Stream cikti = resp.GetResponseStream();

**byte**[] ciktiBytes = **new** **byte**[100];

**int** okunan;

// Stream.Read metodu ile en fazla 100 Byte olacak şekilde çıktı  
// akım nesnesinden bilgi okunmaktadır.

**while** ((okunan = cikti.Read(ciktiBytes, 0, ciktiBytes.Length)) > 0) {

Console.Write(Encoding.UTF8.GetString(ciktiBytes, 0, okunan));

}

VB.NET

**Imports** System.Net

' İstek nesnesini oluştur.

**Dim** istek **As** HttpWebRequest

istek = **CType**(HttpWebRequest.Create("http://www.dotnetturk.com"),   
 HttpWebRequest)

istek.Method = "POST"

' Girdi akım nesnesi alınmakta ve Write metodu ile istenilen bilgi   
' akım nesnesine yazılmaktadır.

**Dim** girdi **As** Stream = istek.GetRequestStream()

**Dim** girdiBytes() **As** **Byte** = Encoding.UTF8.GetBytes("Ara=deneme&Tip=1")

girdi.Write(girdiBytes, 0, girdiBytes.Length)

girdi.Close()

**Dim** resp **As** WebResponse = istek.GetResponse()

**Dim** cikti **As** Stream = resp.GetResponseStream()

**Dim** ciktiBytes() **As** **Byte** = **New** **Byte**(100) {}

**Dim** okunan **As** **Integer** = cikti.Read(ciktiBytes, 0, ciktiBytes.Length)

' Stream.Read metodu ile en fazla 100 Byte olacak şekilde çıktı  
' akım nesnesinden bilgi okunmaktadır.

**While** okunan > 0

Console.Write(Encoding.UTF8.GetString(ciktiBytes, 0, okunan))

okunan = cikti.Read(ciktiBytes, 0, ciktiBytes.Length)

**End** **While**

Kod 207, Örnek Stream.Write İşlemi

Listelenen kaynak kodlar \Bölüm 13\StreamReadWrite klasöründe bulunmaktadır.

Konumlanma ve Akım Pozisyonu

Konumlanma (ing. Seeking), akımın aktif pozisyonunu, okuma veya yazma yapılmak istenilen pozisyon değerine getirebilmeyi ifade etmektedir.

Konumlama işlemi Stream.Position özelliğine istenilen değer atanarak veya Stream.Seek metodu yardımıyla yapılabilir.

| C# | VB.NET |
| --- | --- |
| . . .  // Başlangıç noktasına getir. // Stream.Position = 0 ile aynı // etkiye sahiptir.  Stream.Seek(0, SeekOrigin.Begin);  // Aktif pozisyondan 10 Byte ilerlet. // Stream.Position değerini 10  // artırmak aynı etkiye saiptir. Stream.Seek(10, SeekOrigin.Current);  // Akım sonundan 5 Byte geriye al. // Stream.Position değerini // Stream.Length – 5 değerine // eşitlemekle aynı etkiye sahiptir. Stream.Seek(5, SeekOrigin.End); | . . .  ' Başlangıç noktasına getir. ' Stream.Position = 0 ile aynı ' etkiye sahiptir.  Stream.Seek(0, SeekOrigin.Begin);  ' Aktif pozisyondan 10 Byte ilerlet. ' Stream.Position değerini 10  ' artırmak aynı etkiye saiptir. Stream.Seek(10, SeekOrigin.Current);  ' Akım sonundan 5 Byte geriye al. ' Stream.Position değerini ' Stream.Length – 5 değerine ' eşitlemekle aynı etkiye sahiptir. Stream.Seek(5, SeekOrigin.End); |

Kod 208, Akım Konumlama Örnekleri

Akım nesnesinden veri okuma ve yazma işlemi, akımın o anki pozisyonundan başlayarak yapılmakta, okuma yazma tamamlandıktan sonra akım pozisyonu işlenen veri kadar otomatik olarak artırılmaktadır.  
  
Okuma / yazma sırasında yapılabilecek hatalardan birisi, akım pozisyonunu uygun değere getirmeden okuma ve yazma işlemine başlanmasıdır.

Okuyucu ve Yazıcı Sınıflar

Akım okuyucu ve yazıcı sınıfları, akımlardan veri okuma ve yazma işlevini daha standardize edilmiş yöntemler ile sağlayan sınıflardır.

Bilindiği gibi, Stream.Read ve Stream.Write metotları temel veri okuma ve yazma metotlarıdır. Bu metotlar yardımıyla her tür akım nesnesinden veri okumak / akım nesnesine veri yazmak mümkündür. Okuyucu ve yazıcı sınıflar ile hedeflenen, okuma ve yazma işlevinin daha kolay yapılmasının sağlanmasıdır.

Okuyucu ve yazıcı sınıfları, .NET GÇ mimarisi içerisinde tanımlı olan BinaryReader / BinaryWriter ile StreamReader / StreamReader sınıflarıdır.

BinaryReader / BinaryWriter sınıflarının temel kullanım amacı temel tiplerin ikil bilgiler şeklinde ilgili akım nesnesinden okunabilmesi / ilgili akım nesnesine yazılabilmesinin sağlanabilmesidir.

StreamReader ve StreamWriter sınıfları, metin bazlı verinin akımlardan okunması ve yazılmasını sağlayan sınıflardır. Bu sınıflar sırasıyla TextReader ve TextWriter soyut sınıflarını miras almaktadır.

| C# | VB.NET |
| --- | --- |
| **public** **abstract** **class** TextReader :  MarshalByRefObject,  IDisposable {  }  **public** **abstract** **class** TextWriter :  MarshalByRefObject,  IDisposable {  }  **public** **class** StreamReader :   TextReader {  }  **public** **class** StreamWriter :   TextWriter {  }  **public** **class** BinaryReader :   IDisposable {  }  **public** **class** BinaryWriter :   IDisposable {  } | **Public** **MustInherit** **Class** TextReader  **Inherits** MarshalByRefObject  **Implements** IDisposable  **End** **Class**  **Public** **MustInherit** **Class** TextWriter  **Inherits** MarshalByRefObject  **Implements** IDisposable  **End** **Class**  **Public** **Class** StreamReader  **Inherits** TextReader  **End** **Class**  **Public** **Class** StreamWriter  **Inherits** TextWriter  **End** **Class**  **Public** **Class** BinaryReader  **Implements** IDisposable  **End** **Class**    **Public** **Class** BinaryWriter  **Implements** IDisposable  **End** **Class** |

Kod 209, Akım Okuyucu ve Yazıcı Sınıf Tanımları

İkil Okuyucu – Yazıcı Sınıf Kullanımları

BinaryReader ve BinaryWriter sınıfları, sağlamış oldukları üyeler yardımıyla herhangi bir akım nesnesine temel tip değerlerinin yazılmasını ve okunmasını kolaylıkla yapabilen sınıflardır.

Aşağıda bu sınıflara ait çeşitli üye bilgileri ve açıklamaları gösterilmiştir.

| Üye Adı | Açıklama |
| --- | --- |
| BaseStream | Okuyucu sınıfın kullandığı akım nesnesini döndürür. |
| ReadXXX | Yüklenmiş farklı sürümleri ile temel tiplerin akım nesnesinden okunmasını sağlar. Örneğin  ReadBoolean ReadString ReadInt32 . . . |
| Close | Kullanılan akım nesnesini kapatır. |

Tablo 85, System.IO.BinaryReader Sınıfı Üye Bilgileri

| Üye Adı | Açıklama |
| --- | --- |
| BaseStream | Yazıcı sınıfın kullandığı akım nesnesini döndürür. |
| Write | Yüklenmiş farklı sürümleri ile temel tiplerin akım nesnesine yazılmasını sağlar. Örneğin  Write(true) Write(12)  Write(7.78) . . . |
| Close | Kullanılan akım nesnesini kapatır. |

Tablo 86, System.IO.BinaryWriter Sınıfı Üye Bilgileri

Aşağıdaki kod parçasında, dosya akım nesnesi ve ikil okuyucu / yazıcı sınıfları kullanımı gösterilmiştir. Yazma sırası ile okuma sırasının aynı olması gerektiğine dikkat edilmelidir.

| C# | VB.NET |
| --- | --- |
| // Dosya akım nesnesini yazma modunda // oluştur.  FileStream fs;  fs = **new** FileStream("t.dat",   FileMode.Create);  // İkil yazıcı nesnesini oluştur. // Parametre olarak iletilen akım  // nesnesi, yazma yapılacak akımı // belirtmektedir.  BinaryWriter w;  w = **new** BinaryWriter(fs);  w.Write(123);  w.Write("Ahmet Yılmaz");  w.Write(**new** **byte**[] { 4, 8, 12 });  w.Close();  // Dosya akım nesnesini okuma modunda // oluştur.  fs = **new** FileStream("t.dat",   FileMode.Open);  // Okuyucu nesneyi oluştur. // Parametre olarak iletilen akım  // nesnesi, okuma yapılacak akımı // belirtmektedir.  BinaryReader rd;  rd = **new** BinaryReader(fs);  Console.WriteLine(rd.ReadInt32());  Console.WriteLine(rd.ReadString());  **byte** [] bytes = rd.ReadBytes(3);  **foreach** (**byte** b **in** bytes) {  Console.WriteLine(b);  }  rd.Close(); | ' Dosya akım nesnesini yazma modunda ' oluştur.  **Dim** fs **As** FileStream  fs = **New** FileStream("t.dat", \_  FileMode.Create)  ' İkil yazıcı nesnesini oluştur. ' Parametre olarak iletilen akım  ' nesnesi, yazma yapılacak akımı ' belirtmektedir.  **Dim** w **As** BinaryWriter  w = **New** BinaryWriter(fs)  w.Write(123)  w.Write("Ahmet Yılmaz")  w.Write(**New** **Byte**() {4, 8, 12})  w.Close()  ' Dosya akım nesnesini okuma modunda ' oluştur.  fs = **New** FileStream("t.dat", \_  FileMode.Open)  ' Okuyucu nesneyi oluştur. ' Parametre olarak iletilen akım  ' nesnesi, okuma yapılacak akımı ' belirtmektedir.  **Dim** rd **As** BinaryReader  rd = **New** BinaryReader(fs)  Console.WriteLine(rd.ReadInt32())  Console.WriteLine(rd.ReadString())  **Dim** bytes() **As** **Byte** = rd.ReadBytes(3)  **For** **Each** b As Byte **In** bytes  Console.WriteLine(b)  **Next**  rd.Close() |

Kod 210, Akım İkil Okuyucu / Yazıcı Kullanım Örnekleri

Akım Okuyucu – Yazıcı Kullanım Örnekleri

Metin bazlı verinin akım nesnelerinden okunması / nesnelere yazılması StreamReader ve StreamWriter sınıfları ile kolaylıkla yapılabilmektedir.

Üye bilgileri ve açıklamaları aşağıda gösterilmiştir.

| Üye Adı | Açıklama |
| --- | --- |
| BaseStream | Okuyucu sınıfın kullandığı akım nesnesini döndürür. |
| CurrentEncoding | Aktif veri kodlama standardını belirtir özelliktir. |
| EndOfStream | Akım pozisyonunun akım sonunu işaret etmesi durumunda True değerini döndürür özelliktir. |
| Close | Kullanılan akım nesnesini kapatır. |
| Peek | Akım nesnesinden, akım pozisyonunu etkilemeden bir karakter okur. |
| Read | Akım nesnesinden veri okumak için kullanılmalıdır. |
| ReadLine | Akım nesnesinden bir satır okur ve okuduğu satırı karakter dizi olarak döndürür. |
| ReadToEnd | Akım nesnesinden tüm karakterleri okur ve karakter dizi değeri olarak döndürür. |

Tablo 87, System.IO.StreamReader Sınıfı Üye Bilgileri

| Üye Adı | Açıklama |
| --- | --- |
| AutoFlush | Değerinin True olması, Write metodu çağrımında iletilen verinin akım nesnesine direkt iletilmesi gerektiğini ifade etmektedir. False değere sahip olması, verinin ön bellekte tutulabileceği anlamını taşımaktadır. |
| BaseStream | Yazıcı sınıfın kullandığı akım nesnesini döndürür. |
| Encoding | Aktif veri kodlama standardını belirtir özelliktir. |
| NewLine | Yeni satır değerini döndürür özelliktir. |
| Close | Kullanılan akım nesnesini kapatır. |
| Write | Akım nesnesine veri yazmak için kullanılmalıdır. |
| WriteLine | Akım nesnesine, parametre olarak iletilen değeri ve NewLine özelliği ile belirtilen değeri yazar. |

Tablo 88, System.IO.StreamWriter Sınıfı Üye Bilgileri

StreamReader ve StreamWriter sınıflarının temel amacı, istenilen akım nesnesinden metin bazlı veri okumayı kolaylaştırmalarıdır. Bu sınıflar aynı zamanda dosya işlemleri için de kullanılabilmektedir.

Yapıcı metotlar, herhangi akım nesnesinin yapıcı metoda parametre olarak iletimi veya üzerinde işlem yapılmak istenen dosya yolunun parametre olarak iletimi olmak üzere ikiye gruplanabilir.

İlk kullanım yönteminde, okuma / yazma işleminde ilgili akım nesnesi kullanılmaktadır. İkinci kullanım yönteminde, parametre olarak iletilen dosya yolu ile FileStream nesnesi oluşturulmakta, okuma / yazma işleminde oluşturulan akım nesnesi kullanılmaktadır.

Yapıcı metotlar, her iki kullanım yöntemi için de veri kodlama standardını parametre olarak alabilmektedir.

Yapıcı metot ve üye kullanımı ile ilgili örnek aşağıda gösterilmiştir.

| C# | VB.NET |
| --- | --- |
| // veri.txt isimli dosyayı yazma // amaçlı oluştur.  StreamWriter w;  w = **new** StreamWriter("veri.txt");  // İstenilen değeri dosyaya yazılması.  w.WriteLine(12);  w.WriteLine("12");  w.WriteLine(**true**);  w.WriteLine("test metin ...");  w.Close();  // Benzer şekilde, veri.txt isimli // dosya StreamReader nesnesi ile // açılmaktadır.  StreamReader r1;  r1 = **new** StreamReader("veri.txt");  // Dosya sonuna gelinmediği sürece // konsol penceresine her bir satır  // yazılmaktadır. // Aynı işlem ReadToEnd metodu ile // de yapılabilir.  **while** (!r1.EndOfStream)  Console.WriteLine(r1.ReadLine()); | ' veri.txt isimli dosyayı yazma ' amaçlı oluştur.  **Dim** w **As** StreamWriter  w = **New** StreamWriter("veri.txt")  ' İstenilen değeri dosyaya yazılması.  w.WriteLine(12)  w.WriteLine("12")  w.WriteLine(**True**)  w.WriteLine("test metin ...")  w.Close()  ' Benzer şekilde, veri.txt isimli ' dosya StreamReader nesnesi ile ' açılmaktadır.  **Dim** r1 **As** StreamReader  r1 = **New** StreamReader("veri.txt")  ' Dosya sonuna gelinmediği sürece ' konsol penceresine her bir satır  ' yazılmaktadır. ' Aynı işlem ReadToEnd metodu ile ' de yapılabilir.  **While** **Not** r1.EndOfStream  Console.WriteLine(r1.ReadLine())  **End** **While** |

Kod 211, System.IO.StreamReader ve StreamWriter Kullanım Örnekleri

Dosya Sınıfları

File ve FileInfo isimli sınıflar, benzer işlevlere sahip olan ve dosya sistemi üzerinde işlem yapabilmeyi sağlayan sınıflardır.

Bu sınıfların çeşitli üyeleri StreamReader ve StreamWriter nesnelerini parametre olarak alabilmekte veya bu nesneleri döndürebilmektedir.

Örneğin File.CreateText metodu, parametre olarak iletilen dosya yolu bilgisini kullanarak StreamWriter nesnesi döndürmektedir. Benzer şekilde File.OpenText metodu, ilgili dosya üzerinde okuma işlemi yapabilmek için StreamReader nesnesi döndürmektedir.

Dosya sınıfları ve akım okuyucu – yazıcı sınıfların kullanımı ile ilgili örnek kod parçası aşağıda gösterilmiştir.

| C# | VB.NET |
| --- | --- |
| StreamWriter w;  w = File.CreateText("veri.txt");  *// . . .*  w.Close();  // Dosyayı açar ve yazılan bilginin // dosya sonuna eklenmesini sağlar.  w = File.AppendText("veri.txt");  *// . . .*  w.Close();  StreamReader r;  r = File.OpenText("veri.txt");  *// . . .*  r.Close();  // Aynı işlemin FileInfo sınıfı ile // yapılması da mümkündür.  FileInfo fi;  fi = **new** FileInfo("veri.txt");  r = fi.OpenText(); | **Dim** w **As** StreamWriter  w = File.CreateText("veri.txt")  ' . . .  w.Close()  ' Dosyayı açar ve yazılan bilginin ' dosya sonuna eklenmesini sağlar.  w = File.AppendText("veri.txt")  ' . . .  w.Close()  **Dim** r **As** StreamReader  r = File.OpenText("veri.txt")  ' . . .  r.Close()  ' Aynı işlemin FileInfo sınıfı ile ' yapılması da mümkündür.  **Dim** fi **As** FileInfo = **New** FileInfo("veri.txt")  r = fi.OpenText |

Kod 212, Dosya Sistemi Sınıfları ve Akım Okuyucu – Yazıcı Sınıfları

Dosya Akımları

System.IO.FileStream sınıfı dosya okuma / yazma işlemlerinin yapılabilmesini sağlayan temel dosya akım sınıfıdır.

Yardımcı Tipler

FileStream konusunda detaylı bilgi verilmeden önce, dosyalar üzerinde çalışılırken sık kullanılan bazı tipler hakkında bilgiler sunulacaktır.

FileMode sayılabilir tipi, dosyanın işletim sistemi tarafından nasıl açılması veya oluşturulması gerektiğini belirten tiptir.

| C# | VB.NET |
| --- | --- |
| **public** **enum** FileMode {  // Sonuna ekleme yapmak üzere   // dosyayı aç.  Append = 6,  // Dosya var ise içini boşalt,  // yok ise boş dosya oluştur.  Create = 2,  // Yeni bir dosya oluştur, dosya  // var ise hata ver.  CreateNew = 1,  // Var olan dosyayı aç.  Open = 3,  // Dosya var ise aç, yok ise  // oluştur.  OpenOrCreate = 4,  // Dosya içini boşalt.  Truncate = 5  } | **Public** **Enum** FileMode  ' Sonuna ekleme yapmak üzere   ' dosyayı aç.  Append = 6  ' Dosya var ise içini boşalt,  ' yok ise boş dosya oluştur.  Create = 2  ' Yeni bir dosya oluştur, dosya  ' var ise hata ver.  CreateNew = 1  ' Var olan dosyayı aç.  Open = 3  ' Var olan dosyayı aç.  ' Dosya var ise aç, yok ise  ' oluştur.  OpenOrCreate = 4  ' Dosya içini boşalt.  Truncate = 5  **End** **Enum** |

Kod 213, System.IO.FileMode Tip Tanımı

FileAccess sayılabilir tipi, dosya üzerinde yapılacak işlemi belirtir tiptir.

| C# | VB.NET |
| --- | --- |
| **public** **enum** FileAccess {  // Sadece okuma  Read = 1,  // Okuma ve Yazma  ReadWrite = 3,  // Sadece yazma  Write = 2  } | **Public** **Enum** FileAccess  ' Sadece okuma  Read = 1  ' Okuma ve Yazma  ReadWrite = 3  ' Sadece yazma  Write = 2  **End** **Enum** |

Kod 214, System.IO.FileAccess Tip Tanımı

Son olarak FileShare sayılabilir tipi, dosyanın farklı FileStream nesneleri tarafından açılması, okunması veya yazılmasının ne şekilde düzenlenmesi gerektiğini belirtmektedir. Örneğin dosya, bir kullanıcı veya program tarafından açılmış ve kullanılıyor iken, bir başka kullanıcı veya program aynı dosyaya erişmek isteyebilir.

| C# | VB.NET |
| --- | --- |
| **public** **enum** FileShare {  // Silme iznini belirtir.  Delete = 4,    // Direkt desteklenmemektedir.  Inheritable = 0x10,  // Paylaşımı engeller.  None = 0,  // Sadece okuma izni.  Read = 1,  // Okuma ve yazma izni.  ReadWrite = 3,  // Sadece yazma izni.  Write = 2  } | **Public** **Enum** FileShare  ' Silme iznini belirtir.  Delete = 4  ' Direkt desteklenmemektedir.  Inheritable = &H10  ' Paylaşımı engeller.  None = 0  ' Sadece okuma izni.  Read = 1  ' Okuma ve yazma izni.  ReadWrite = 3  ' Sadece yazma izni.  Write = 2  **End** **Enum** |

Kod 215, System.IO.FileShare Tip Tanımı

Yapıcı Metot Bilgileri

FileStream sınıfı, yüklenmiş sürümleri bulunan oldukça zengin sayıda yapıcı metotlar tanımlamaktadır.

Aşağıda en sık kullanılan ve önemli görülen yapıcı metot tanım ve açıklamaları gösterilmiştir.

| C# | VB.NET |
| --- | --- |
| FileStream(**string** path,  FileMode mode,  FileAccess access) {  }  FileStream(**string** path,  FileMode mode,  FileAccess access,  FileShare share) {  } | **Sub** **New**(**ByVal** path **As** **String**, \_  **ByVal** mode **As** FileMode, \_  **ByVal** access **As** FileAccess)  **End** **Sub**  **Sub** **New**(**ByVal** path **As** **String**, \_  **ByVal** mode **As** FileMode, \_  **ByVal** access **As** FileAccess, \_  **ByVal** share **As** FileShare)  **End** **Sub** |

Kod 216, System.IO.FileStream Yapıcı Metot Bilgileri

Üye Bilgileri ve Kullanım Örnekleri

FileStream sınıfı üye bilgi ve açıklamaları aşağıda gösterilmiştir. Bilindiği gibi FileStream, Stream sınıfını miras almaktadır. Stream tarafından tanımlanan bir çok soyut ve sanal metot, amaçlarına uygun olarak, FileStream sınıfı tarafından örtülmüştür.

| Üye Adı | Açıklama |
| --- | --- |
| Handle | İşletim sistemi tarafından yönetilen değeri döndürür özelliktir. |
| Name | Dosya adını döndürür özelliktir. |
| Lock | Belirtilen pozisyon ve uzunluktaki dosya bölümünü dış erişime kapatır. |
| Unlock | Belirtilen pozisyon ve uzunluktaki dosya bölümünü dış erişime açar. |

Tablo 89, System.IO.FileStream Sınıfı Üye Bilgileri

Aşağıda temel kullanım örneği gösterilmiştir.

C#

FileStream f = **new** FileStream("c:\\veri.dat", FileMode.Create);

// Dosya boyutunu 1000 Byte olarak belirle.

f.SetLength(1000);

// Dosyaya yaz.

**byte**[] veri = Encoding.UTF8.GetBytes("DotNet");

f.Write(veri, 0, veri.Length);

f.Close();

f = **new** FileStream("c:\\veri.dat", FileMode.Open);

Console.WriteLine(f.Length);

// 1000

// Dosyayı sonuna kadar oku.

StreamReader r = **new** StreamReader(f);

Console.WriteLine(r.ReadToEnd());

r.Close();

VB.NET

**Dim** f **As** FileStream = **New** FileStream("c:\\veri.dat", FileMode.Create)

' Dosya boyutunu 1000 Byte olarak belirle.

f.SetLength(1000)

' Dosyaya yaz.

**Dim** veri() **As** **Byte** = Encoding.UTF8.GetBytes("DotNet")

f.Write(veri, 0, veri.Length)

f.Close()

f = **New** FileStream("c:\\veri.dat", FileMode.Open)

Console.WriteLine(f.Length)

' 1000

' Dosyayı sonuna kadar oku.

**Dim** r **As** StreamReader = **New** StreamReader(f)

Console.WriteLine(r.ReadToEnd())

r.Close()

Kod 217, System.IO.FileStream Sınıfı Kullanım Örneği

Hafıza Akımları

System.IO.MemoryStream sınıfı hafıza okuma / yazma işlemlerinin yapılabilmesini sağlayan temel hafıza akım sınıfıdır.

Hafıza akımları ile hedeflenen, verinin direkt bilgisayar hafızasında işlenebilmesinin sağlanmasıdır.

Yapıcı Metot Bilgileri

MemoryStream yapıcı metot tanım ve açıklamaları aşağıda gösterilmiştir.

| C# | VB.NET |
| --- | --- |
| MemoryStream() {  }  // İlk kapasite değerinin iletildiği // yapıcı metot.  MemoryStream(**int** capacity) {  }  // Hafıza akım nesnesi ilk içeriğinin // iletildiği yapıcı metot.  MemoryStream(**byte**[] buffer) {  }  // Hafıza akım nesnesi ilk içeriğinin // iletildiği yapıcı metot. writable // parametresine false değeri atanması // akım nesnesinin salt okunur  // olmasını sağlamaktadır.  MemoryStream(**byte**[] buffer,  **bool** writable) {  } | **Sub** **New**()  **End** **Sub**  ' İlk kapasite değerinin iletildiği ' yapıcı metot.  **Sub** **New**(**ByVal** capacity **As** **Integer**)  **End** **Sub**  ' Hafıza akım nesnesi ilk içeriğinin ' iletildiği yapıcı metot.  **Sub** **New**(**ByVal** buffer **As** **Byte**())  **End** **Sub**  ' Hafıza akım nesnesi ilk içeriğinin ' iletildiği yapıcı metot. writable ' parametresine False değeri atanması ' akım nesnesinin salt okunur  ' olmasını sağlamaktadır.  **Sub** **New**(**ByVal** buffer **As** **Byte**(), \_  **ByVal** writable **As** **Boolean**)  **End** **Sub** |

Kod 218, System.IO.MemoryStream Yapıcı Metot Bilgileri

Üye Bilgileri ve Kullanım Örnekleri

MemoryStream sınıfı üye bilgi ve açıklamaları aşağıda gösterilmiştir.

| Üye Adı | Açıklama |
| --- | --- |
| GetBuffer | Akım nesnesi oluşturulurken parametre olarak iletilen byte diziyi döndürür. |
| ToArray | Akım içeriğini byte dizi olarak döndürür. |
| WriteTo | Parametre olarak iletilen akım nesnesine akım içeriğini yazar. |

Tablo 90, System.IO.MemoryStream Sınıfı Üye Bilgileri

C#

// Hafıza akım nesnesi oluştur.

MemoryStream ms;

ms = **new** MemoryStream();

// Akım yazıcı nesnesi oluştur.

StreamWriter w = **new** StreamWriter(ms, Encoding.UTF32);

w.WriteLine("DotNet Türk");

w.Flush();

Console.WriteLine(ms.Length);

// 56

**byte**[] sonuc = ms.ToArray();

FileStream f = **new** FileStream("c:\\veri.dat", FileMode.Create);

// Hafıza akım nesnesi içeriğini dosya akım nesnesine yaz.

ms.WriteTo(f);

ms.Close();

f.Close();

VB.NET

' Hafıza akım nesnesi oluştur.

**Dim** ms **As** MemoryStream = **New** MemoryStream()

' Akım yazıcı nesnesi oluştur.

**Dim** w **As** StreamWriter= **New** StreamWriter(ms, Encoding.UTF32)

w.WriteLine("DotNet Türk")

w.Flush()

Console.WriteLine(ms.Length)

' 56

**Dim** sonuc() **As** **Byte** = ms.ToArray()

**Dim** f **As** FileStream= **New** FileStream("c:\\veri.dat", FileMode.Create)

' Hafıza akım nesnesi içeriğini dosya akım nesnesine yaz.

ms.WriteTo(f)

ms.Close()

f.Close()

Kod 219, System.IO.MemoryStream Sınıfı Kullanım Örneği

Ön Bellek Akımları

Ön bellek akımları, bir başka akım nesnesi üzerinde yapılan okuma ve yazma işlemlerinde, yeni bir katman oluşturarak verinin ön bellekte tutulabilmesini sağlamaktadır.

System.IO.BufferedStream sınıfı temel ön bellek akım sınıfıdır. BufferedStream sınıfı, arka planda yönettiği ön bellek üzerinde yazma ve okuma işlemlerini yapmakta, gerektiği durumda işlemleri gerçek akım nesnesine yansıtmaktadır.

Aşağıda yapıcı metot tanımları gösterilmiştir.

| C# | VB.NET |
| --- | --- |
| // stream parametresi ile belirtilen // değer, ön bellek akımı // tarafından kullanılması istenen // akım nesnesini belirtmektedir.  BufferedStream(Stream stream) {  }  // Akım nesnesi ve ön bellek boyutunu // parametre olarak alan yapıcı // metot.  BufferedStream(Stream stream,   **int** bufferSize) {  } | ' stream parametresi ile belirtilen ' değer, ön bellek akımı ' tarafından kullanılması istenen ' akım nesnesini belirtmektedir.  **Sub** **New**(**ByVal** stream **As** Stream)  **End** **Sub**  ' Akım nesnesi ve ön bellek boyutunu ' parametre olarak alan yapıcı ' metot.  **Sub** **New**(**ByVal** stream **As** Stream, \_  **ByVal** bufferSize **As** **Integer**)  **End** **Sub** |

Kod 220, System.IO.BufferedStream Sınıfı Yapıcı Metotları

Veri Sıkıştırma

Veri sıkıştırma (ing. Data Compression), mevcut verinin daha az yer kaplayacak şekilde kodlanması olarak tanımlanabilir.

Veri Sıkıştırma Algoritmaları

Veri sıkıştırma amacıyla geliştirilmiş farklı algoritmalar bulunmaktadır. Bu konu başlığında en sık kullanılan GZip ve Deflate veri sıkıştırma algoritmaları ve bu algoritmaların .NET içerisinde kullanımları hakkında bilgi verilecektir.

GZip ve Deflate, aynı veri sıkıştırma algoritmasını kullanmalarına rağmen, GZip, Deflate'ten farklı olarak, çeşitli bilgiler içerebilen başlık verisini desteklemektedir.

Veri Sıkıştırma Akımları

System.IO.Compression ad alanı içerisinde tanımlı olan GZipStream ve DeflateStream sınıfları, sırasıyla GZip ve Deflate veri sıkıştırma algoritmalarının kolaylıkla kullanılabileceği akım sınıflarıdır.

Aşağıda DeflateStream nesneleri kullanılarak bir dosyanın sıkıştırılması ve sıkıştırılmış dosyanın açılması gösterilmiştir.

C#

// veri.dat isimli dosyanın veri.def olarak sıkıştırılması ...

DeflateStream defStream;

// veri.dat dosyasını aç.

FileStream f = **new** FileStream("veri.dat", FileMode.Open);

// veri.def dosyasını oluştur.

FileStream d = **new** FileStream("veri.def", FileMode.Create);

// DeflateStream nesnesini veri sıkıştırma modunda oluştur.

defStream = **new** DeflateStream(d, CompressionMode.Compress);

**byte**[] buffer = **new** **byte**[1000];

**int** boyut;

// veri.dat isimli dosyadan oku ve okunan bilgileri DeflateStream  
// akım nesnesine yaz.

**while** ((boyut = f.Read(buffer, 0, buffer.Length)) > 0) {

defStream.Write(buffer, 0, boyut);

}

defStream.Close();

f.Close();

d.Close();

// veri.def isimli dosyanın veri.dat olarak açılması . . .

// veri.dat dosyasını oluştur.

d = **new** FileStream("veri.dat", FileMode.Create);

// veri.def dosyasını aç.

f = **new** FileStream("veri.def", FileMode.Open);

// DeflateStream nesnesini sıkıştırılmış veriyi açma modunda oluştur.

defStream = **new** DeflateStream(f, CompressionMode.Decompress);

// Açılmış veriyi oku ve yaz.

**while** ((boyut = defStream.Read(buffer, 0, buffer.Length)) > 0) {

d.Write(buffer, 0, boyut);

}

defStream.Close();

f.Close();

d.Close();

VB.NET

' veri.dat isimli dosyanın veri.def olarak sıkıştırılması ...

**Dim** defStream **As** DeflateStream

' veri.dat dosyasını aç.

**Dim** f **As** FileStream = **New** FileStream("veri.dat", FileMode.Open)

' veri.def dosyasını oluştur.

**Dim** d **As** FileStream = **New** FileStream("veri.def", FileMode.Create)

' DeflateStream nesnesini veri sıkıştırma modunda oluştur.

defStream = **New** DeflateStream(d, CompressionMode.Compress)

**Dim** buffer(1000) **As** **Byte**

' veri.dat isimli dosyadan oku ve okunan bilgileri DeflateStream  
' akım nesnesine yaz.

**Dim** boyut **As** **Integer** = f.Read(buffer, 0, buffer.Length)

**While** boyut > 0

defStream.Write(buffer, 0, boyut)

boyut = f.Read(buffer, 0, buffer.Length)

**End** **While**

defStream.Close()

f.Close()

d.Close()

' veri.def isimli dosyanın veri.dat olarak açılması . . .

' veri.dat dosyasını oluştur.

d = **New** FileStream("veri.dat", FileMode.Create)

' veri.def dosyasını aç.

f = **New** FileStream("veri.def", FileMode.Open)

' DeflateStream nesnesini veri sıkıştırma modunda oluştur.

defStream = **New** DeflateStream(f, CompressionMode.Decompress)

' Açılmış veriyi oku ve yaz.

boyut = defStream.Read(buffer, 0, buffer.Length)

**While** boyut > 0

d.Write(buffer, 0, boyut)

boyut = defStream.Read(buffer, 0, buffer.Length)

**End** **While**

defStream.Close()

f.Close()

d.Close()

Kod 221, System.IO.Compression.DeflateStream Kullanım Örneği

Hikmetli bilgi, tecrübe ile desteklenmiş ve uygulanabilir özellikler taşıyan ilimdir. Hikmet, ilim ile sanatın birleşmesidir.

Elmalılı M.HamdiYazır

Bölüm

14

Nesne Serilizasyonu

S

erilizasyon, nesnelerin istenilen veri yapısında depolanmasına / transfer edilmesine verilen isimdir. De-serilizasyon, depolanmış / transfer edilmiş veriden tekrar nesnenin oluşturulmasıdır.

Kitabın bu bölümünde nesne serilizasyonu ve de-serilizasyon konularında temel bilgiler verilecektir.

Serilizasyon ve de-serilizasyon; kullanım, özelleştirilebilme ve güvenlik yönleri ile oldukça kapsamlı ve detaylı bir konudur. Bu bölümde, pratikte serilizasyon ve de-serilizasyon kullanımı ile ilgili temel düzey açıklamalar yapılacaktır.

Temel Kavramlar

Nesneleri 'Sınıf Örnekleri' olarak tanımlamak mümkündür. Bir nesne new kelimesi ile oluşturulduğunda, oluşturulan nesne bilgisayar hafızasında saklanmaktadır.

Serilizasyon ile hedeflenen, bilgisayar hafızasında saklanan nesnenin, akımlar yardımı ile, istenilen ortamda saklanabilmesinin sağlanmasıdır. Benzer şekilde de-serilizasyon, farklı ortamda saklanan bir nesnenin tekrar bilgisayar hafızasında oluşturulmasıdır.

Serilizasyonu nesnenin dönüşümü olarak algılamak yanlış olmayacaktır. Serilizasyon işlemi tamamlandıktan sonra nesne, farklı bir veri yapısına dönüşmektedir. De-serilizasyon ise, farklı veri yapısından nesnenin yeniden oluşturulması (new kelimesi kullanmadan) olarak değerlendirilebilir.

Bir nesnenin serilize edilmesi, farklı formatlayıcılar yardımıyla serilizasyon işleminin yapılması, nesnenin dönüştürülmesidir. Nesnenin de-serilize edilmesi, nesnenin yeniden oluşturulmasını ifade etmektedir.

Serilize Edilebilirlik ve Formatlayıcılar

Bir nesnenin serilize edilebilirliği Serializable özniteliği ile yönetilmektedir. Nesne üzerinde serilizasyonun yapılabilmesi için ilgili nesnenin Serializable öz niteliği ile donatılmış olması gerekmektedir. Serializable özniteliği ile donatılmayan nesnelerin .NET serilizasyon mimarisi ile serilize edilmeleri mümkün değildir.

Formatlayıcı, (ing. Formatter) serilizasyon ve de-serilizasyon işleminde .NET serilizasyon mimarisi tarafından üretilen veriyi formatlayan sınıflara verilen genel isimdir.

Formatlayıcılar System.Runtime.Serialization.IFormatter arabirim gerçekleştirimini yapmaktadır.

.NET Serilizasyon Mimarisi

.NET serilizasyon mimarisi, nesnelerin serilizasyon ve de-serilizasyon işlemlerini yöneten, güvenlik ve özelleştirilebilme gibi konularda gerekli desteği sağlayan mimaridir.

Serilizasyon mimarisi tip tanımlarının çoğunluğu System.Runtime.Serialization ad alanı içerisinde toplanmıştır. Buna rağmen bazı formatlayıcı sınıfları, farklı ad alanı ve Assembly içerisinde bulunmaktadır.

Serilizasyon mimarisi, sağladıkları işlevler aynı olmakla birlikte veri yapısı ve kullanım özellikleri değişebilen, üç temel formatlayıcı sınıf içermektedir.

| Sınıf | Açıklama |
| --- | --- |
| BinaryFormatter | Serilize ettiği nesneyi ikil veri yapısında saklayan sınıftır. |
| SoapFormatter | Serilize ettiği nesneyi Soap (Simple Object Access Protocol) yapısında saklayan sınıftır. |
| XmlSerializer | Serilize ettiği nesneyi Xml (Extensible Markup Language) yapısında saklayan sınıftır. |

Tablo 91, Serilizasyon Sınıf ve Açıklamaları

BinaryFormatter sınıfı, serilize ettiği nesneyi ikil veri yapısında daha az yer kaplayacak şekilde saklayabilmektedir. Serilize edilmiş veriye sadece .NET uygulamaları tarafından erişileceği durumlarda bu formatlayıcı tercih edilmelidir.

SoapFormatter sınıfı, web hizmetlerinde (ing. Web Services) kullanılan Soap yapısını desteklemektedir. Bu formatlayıcı ile serilize edilmiş nesneler, metin veri yapısında ve Xml benzeri şekilde saklanmakta / iletilmektedir. SoapFormatter, pratikte, sadece web hizmetlerinde ASP.NET tarafından kullanılmaktadır.

XmlSerializer sınıfı, Xml yapısında serilizasyonu desteklemektedir. Bu sınıf ile elde edilen en önemli avantaj, tamamen açık ve bağımsız bir veri yapısı sunan Xml yardımıyla, serilize edilmiş nesnelerin rahatlıkla farklı platform ve uygulamalar tarafından da erişilebilir olmasının sağlanmasıdır.

XML ve SOAP konusunun detayları bu kitabın kapsamı dışındadır. http://www.oasis-open.org internet sitesinde bu yapılar ile ilgili detaylı bilgi bulunabilir.  
  
SoapFormatter sınıfı System.Runtime.Serialization.Formatters.Soap.dll içerisinde, XmlSerializer sınıfı System.Xml.dll içerisinde bulunmaktadır.  
  
Formatlayıcı sınıflar direkt object sınıfını miras almaktadır. Formatlayıcı sınıfların ortak özelliği IFormatter arabirim gerçekleştirimini yapmalarıdır. Programcıların, bu arabirim gerçekleştirimini yaparak istedikleri yapıda veriyi formatlayan sınıflar geliştirmeleri mümkündür.  
  
XmlSerializer sınıfı serilizasyon ve de-serilizasyon yapabilmesine rağmen IFormatter arabirim gerçekleştirimini yapmamaktadır.

Örnek Serilizasyon ve De-Serilizasyon

Birçok Windows uygulama yazılımında kullanıcı ayarlarının saklanması ihtiyacı vardır. Kullanıcı ayarları, genellikle kullanıcının bilgisayarında kayıt defteri (ing. Registry) veya herhangi bir dosya içerisinde saklanabilir.

Kullanıcı ayarlarının yazılım içerisinde bir sınıf ile ifade edilebileceği değerlendirildiğinde, aşağıda bu amaçla kullanılabilecek – basitleştirilmiş – örnek bir sınıf gösterilmiştir.

C#

**class** KullanıcıAyarları {

**public** **int** FormUstPozisyon;

**public** **int** FormSolPozisyon;

**public** **bool** BeniHatırla;

}

VB.NET

**Class** KullanıcıAyarları

**Public** FormUstPozisyon **As** **Integer**

**Public** FormSolPozisyon **As** **Integer**

**Public** BeniHatırla **As** **Boolean**

**End** **Class**

Kod 222, Örnek Kullanıcı Ayarları Sınıfı

Uygulama açılırken bu sınıfa ait nesne oluşturulabilir ve alan değerleri dosyadan okunabilir, uygulama kapanırken ise mevcut nesne alanları dosyaya kaydedilebilir.

C#

KullanıcıAyarları ayarlar = **new** KullanıcıAyarları();

FileStream fs;

// Kullanıcı ayarları nesnesi alan değerleri ayarlar.dat isimli dosyadan  
// okunmaktadır.

**if** (File.Exists("ayarlar.dat")) {

fs = **new** FileStream("ayarlar.dat", FileMode.Open);

BinaryReader rd = **new** BinaryReader(fs);

ayarlar.FormUstPozisyon = rd.ReadInt32();

ayarlar.FormSolPozisyon = rd.ReadInt32();

ayarlar.BeniHatırla = rd.ReadBoolean();

rd.Close();

}

// . . .

// Kullanıcı programı açmakta ve kullanıcıya ait ayarlar değişmektedir.

// . . .

// Değişmiş olan kullanıcı ayarları nesnesi alan değerleri dosyaya   
// yazılmaktadır.

fs = **new** FileStream("ayarlar.dat", FileMode.Create);

BinaryWriter bw = **new** BinaryWriter(fs);

bw.Write(ayarlar.FormUstPozisyon);

bw.Write(ayarlar.FormSolPozisyon);

bw.Write(ayarlar.BeniHatırla);

bw.Close();

VB.NET

**Dim** ayarlar **As** KullanıcıAyarları = **New** KullanıcıAyarları()

**Dim** fs **As** FileStream

' Kullanıcı ayarları nesnesi alan değerleri ayarlar.dat isimli dosyadan  
' okunmaktadır.

**If** File.Exists("ayarlar.dat") **Then**

fs = **New** FileStream("ayarlar.dat", FileMode.Open)

**Dim** rd **As** BinaryReader = **New** BinaryReader(fs)

ayarlar.FormUstPozisyon = rd.ReadInt32()

ayarlar.FormSolPozisyon = rd.ReadInt32()

ayarlar.BeniHatırla = rd.ReadBoolean()

rd.Close()

**End** **If**

' . . .

' Kullanıcı programı açmakta ve kullanıcıya ait ayarlar değişmektedir.

' . . .

' Değişmiş olan kullanıcı ayarları nesnesi alan değerleri dosyaya   
' yazılmaktadır.

fs = **New** FileStream("ayarlar.dat", FileMode.Create)

**Dim** bw **As** BinaryWriter = **New** BinaryWriter(fs)

bw.Write(ayarlar.FormUstPozisyon)

bw.Write(ayarlar.FormSolPozisyon)

bw.Write(ayarlar.BeniHatırla)

bw.Close()

Kod 223, Örnek Kullanıcı Ayarları Nesnesi Okuma ve Yazma

Yukarıdaki örnek kod parçasında, en basit şekliyle nesne serilizasyon ve de-serilizasyon işleminin programcının kendisi tarafından yapılması örneklenmiştir.

Uygulama çalıştığında ilk adım olan de-serilizasyon işleminde, ayarlar.dat isimli dosya içerisinden nesne alan değerleri okunmaktadır. Uygulama kullanıcı tarafından kapatılmadan hemen önce serilizasyon işlemi ile nesne alan değerleri aynı dosyaya yazılmaktadır. Böylece, kullanıcı ayarları disk sistemi üzerindeki bir dosyada saklanmaktadır.

Aşağıdaki alt konu başlıklarında, programcı tarafından yapılan basit serilizasyon ve de-serilizasyon işleminin .NET serilizasyon mimarisi sınıfları ile yapılması aktarılmıştır.

İkil Serilizasyon Örneği

.NET serilizasyon sınıfları ile nesne serilizasyonun yapılabilmesi için öncelikle ilgili nesnenin Serializable öz niteliği ile donatılması gerekmektedir.

C#

[Serializable]

**class** KullanıcıAyarları {

**public** **int** FormUstPozisyon;

**public** **int** FormSolPozisyon;

**public** **bool** BeniHatırla;

}

VB.NET

<Serializable> \_

**Class** KullanıcıAyarları

**Public** FormUstPozisyon **As** **Integer**

**Public** FormSolPozisyon **As** **Integer**

**Public** BeniHatırla **As** **Boolean**

**End** **Class**

BinaryFormatter kullanılarak yapılan serilizasyon ve de-serilizasyon işlemi aşağıda gösterilmiştir.

| C# | VB.NET |
| --- | --- |
| // Serilize edilecek olan nesne // oluşturulmaktadır.  KullanıcıAyarları ayarlar;  ayarlar = **new** KullanıcıAyarları();  ayarlar.FormUstPozisyon = 12;  ayarlar.FormSolPozisyon = 34;  ayarlar.BeniHatırla = **true**;  BinaryFormatter bf;  bf = **new** BinaryFormatter();  // ayarlar.bin isimli dosya  // FileStream nesnesi ile  // oluşturulmaktadır.  FileStream fs;  fs = **new** FileStream("ayarlar.bin",  FileMode.Create);  // Serialize metodu herhangi bir akım // nesnesi ve serilize edilmek istenen // nesneyi parametre olarak  // almaktadır.  bf.Serialize(fs, ayarlar);  fs.Close();  fs = **new** FileStream("ayarlar.bin",  FileMode.Open);  // Deserialize metodu herhangi bir  // akım nesnesini parametre olarak // almaktadır. Deserilizasyonun // başarılı olması durumunda  // de-serilize edilmiş olan nesne // bu metot tarafından  // döndürülmektedir.  object o = bf.Deserialize(fs);  // Tip dönüşüm kuralları ile nesne // KullanıcıAyarları tipine  // dönüştürülmektedir.  KullanıcıAyarları ayarlar2;  ayarlar2 = (KullanıcıAyarları)o;  fs.Close(); | ' Serilize edilecek olan nesne ' oluşturulmaktadır.  **Dim** ayarlar **As** KullanıcıAyarları  ayarlar = **New** KullanıcıAyarları()  ayarlar.FormUstPozisyon = 12  ayarlar.FormSolPozisyon = 34  ayarlar.BeniHatırla = **True**  **Dim** bf **As** BinaryFormatter  bf = **New** BinaryFormatter()  ' ayarlar.bin isimli dosya FileStream ' nesnesi ile oluşturulmaktadır.  **Dim** fs **As** FileStream  fs = **New** FileStream("ayarlar.bin", \_  FileMode.Create)  ' Serialize metodu herhangi bir akım ' nesnesi ve serilize edilmek istenen ' nesneyi parametre olarak  ' almaktadır.  bf.Serialize(fs, ayarlar)  fs.Close()  fs = **New** FileStream("ayarlar.bin",  FileMode.Open)  ' Deserialize metodu herhangi bir  ' akım nesnesini parametre olarak ' almaktadır. Deserilizasyonun ' başarılı olması durumunda  ' de-serilize edilmiş olan nesne ' bu metot tarafından  ' döndürülmektedir.  **Dim** o **As** **Object** = bf.Deserialize(fs)  ' Tip dönüşüm kuralları ile nesne ' KullanıcıAyarları tipine  ' dönüştürülmektedir.  **Dim** ayarlar2 **As** KullanıcıAyarları  ayarlar2 = **CType**(o, KullanıcıAyarları)  fs.Close() |

Kod 224, İkil Formatlayıcı ile Örnek Serilizasyon İşlemi

Ayarlar.bin isimli dosya içeriği aşağıda gösterilmiştir.

ÿÿÿÿJConsoleApplication, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" KullanÄ±cÄ±AyarlarÄ±FormUstPozisyonFormSolPozisyonBeniHatÄ±rla"

Soap Serilizasyon Örneği

Önceki konuda BinaryFormatter sınıfı ile yapılan ikil serilizasyon işleminin SoapFormatter sınıfı ile yapılması aşağıda gösterilmiştir.

| C# | VB.NET |
| --- | --- |
| . . .  SoapFormatter sf;  sf = **new** SoapFormatter();  FileStream fs;  fs = **new** FileStream("ayarlar.soap",   FileMode.Create);  sf.Serialize(fs, ayarlar);  fs.Close();  fs = **new** FileStream("ayarlar.soap",   FileMode.Open);  **object** o = sf.Deserialize(fs);  . . . | . . .  **Dim** sf **As** SoapFormatter  sf = **New** SoapFormatter()  **Dim** fs **As** FileStream  fs = **New** FileStream("ayarlar.bin", \_  FileMode.Create)  sf.Serialize(fs, ayarlar)  fs.Close()  fs = **New** FileStream("ayarlar.bin", \_  FileMode.Open)  **Dim** o **As** **Object** = sf.Deserialize(fs)  . . . |

Kod 225, Soap Formatlayıcı ile Örnek Serilizasyon İşlemi

Ayarlar.soap isimli dosya içeriği aşağıda gösterilmiştir.

<SOAP-ENV:Envelope xmlns:xsi=". . .">

<SOAP-ENV:Body>

<a1:KullanıcıAyarları id="ref-1" xmlns:a1="http://schemas.microsoft.com/clr/nsassem/ConsoleApplication%2C%20Version%3D1.0.0.0%2C%20Culture%3Dneutral%2C%20PublicKeyToken%3Dnull">

<FormUstPozisyon>12</FormUstPozisyon>

<FormSolPozisyon>34</FormSolPozisyon>

<BeniHatırla>true</BeniHatırla>

</a1:KullanıcıAyarları>

</SOAP-ENV:Body>

</SOAP-ENV:Envelope>

XML Serilizasyon Örneği

Serilizasyonun XmlSerializer sınıfı ile yapılması aşağıda gösterilmiştir.

C#

// XmlSerializer nesnesi oluşturulmaktadır. Serilize edilmek istenen nesne   
// tipinin de parametre olarak yapıcı metoda iletildiğine dikkat edilmelidir.

XmlSerializer xf = **new** XmlSerializer(**typeof**(KullanıcıAyarları));

FileStream fs = **new** FileStream("ayarlar.xml", FileMode.Create);

xf.Serialize(fs, ayarlar);

fs.Close();

fs = **new** FileStream("ayarlar.xml", FileMode.Open);

**object** o = xf.Deserialize(fs);

VB.NET

' XmlSerializer nesnesi oluşturulmaktadır. Serilize edilmek istenen nesne   
' tipinin de parametre olarak yapıcı metoda iletildiğine dikkat edilmelidir.

**Dim** xf **As** XmlSerializer = **New** XmlSerializer(**GetType**(KullanıcıAyarları))

**Dim** fs **As** FileStream = **New** FileStream("ayarlar.xml", FileMode.Create)

xf.Serialize(fs, ayarlar)

fs.Close()

fs = **New** FileStream("ayarlar.xml", FileMode.Open)

**Dim** o **As** **Object** = xf.Deserialize(fs)

Kod 226, Xml Serilizasyon İşlemi

Ayarlar.xml isimli dosya içeriği aşağıda gösterilmiştir.

<?xml version="1.0"?>

<KullanıcıAyarları

xmlns:xsi=". . ."

xmlns:xsd=". . .">

<FormUstPozisyon>12</FormUstPozisyon>

<FormSolPozisyon>34</FormSolPozisyon>

<BeniHatırla>true</BeniHatırla>

</KullanıcıAyarları>

Çoğu internet tarayıcısı XML yapısında olan dosyaları görüntüleyebilmektedir.

Formatlayıcı Seçim Önerileri

Serilize edilmiş nesnelerin sadece .NET uygulamaları tarafından de-serilize edilmesi gerekiyorsa en hızlı formatlayıcı olan BinaryFormatter kullanımı önerilmektedir.

SoapFormatter çoğunlukla web hizmetleri için kullanılmaktadır.

XmlSerializer sınıfı, verinin farklı platform ve .NET dışındaki uygulamalar tarafından işlenmesi gerekiyorsa tercih edilmelidir. Düz metin yapısında serilizasyon yapan bu sınıf nesnelerin Xml dokümanlarına dönüşümü ve transferi için kullanılabilmektedir.

Serilizasyonun Özelleştirilmesi

.NET tarafından sunulan serilizasyon mimarisi çoğu durumda ihtiyaçları birebir karşılamaktadır. Buna rağmen, serilizasyon / de-serilizasyon sürecine programcı tarafından müdahale edilmesi ve süreçte çeşitli değişiklikler yapılabilmesi mümkündür.

Örneğin ön tanımlı serilizasyon, nesnenin bütün alanlarını nesne ile birlikte serilize etmektedir. Bazı durumlarda sadece istenen alanların serilize edilmesi veya serilizasyon başlamadan önce / tamamlandıktan sonra işlemler yapılması gerekebilir. Bu ve benzeri durumlarda süreç programcı tarafından yönetilebilir.

System.NonSerializedAttribute öz niteliği ile nesne üyesinin donatılması, ilgili üyenin serilize edilmemesi gerektiğini belirtmektedir.

| C# | VB.NET |
| --- | --- |
| [Serializable]  **class** KullanıcıAyarları {  **public** **int** FormUstPozisyon;  **public** **int** FormSolPozisyon;  [NonSerialized]  **public** DateTime SonGirisTarihi;  **public** **bool** BeniHatırla;  } | <Serializable> **Class** KullanıcıAyarları  **Public** FormUstPozisyon **As** **Integer**  **Public** FormSolPozisyon **As** **Integer**  <NonSerialized()> \_  **Public** SonGirisTarihi **As** DateTime  **Public** BeniHatırla **As** **Boolean**  **End** **Class** |

Kod 227, System.NonSerializedAttribute Kullanım Örneği

Yukarıdaki örnekte SonGirisTarihi isimli alan, BinaryFormatter veya SoapFormatter ile yapılan serilizasyonda göz ardı edilecektir.

NonSerialized öz niteliği XmlSerializer için geçerli değildir. XmlSerializer için aynı işlev XmlIgnore öz niteliği ile sağlanabilir.

ISerializable Arabirimi

System.Runtime.Serialization.ISerializable arabirimi, serilizasyon sürecinin özelleştirilmesi amacıyla kullanılabilir.

C#

**public** **interface** ISerializable {

**void** GetObjectData(SerializationInfo info, StreamingContext context);

}

VB.NET

**Public** **Interface** ISerializable

**Sub** GetObjectData(**ByVal** info **As** SerializationInfo, \_  
 **ByVal** context **As** StreamingContext)

**End** **Interface**

Kod 228, System.Runtime.Serialization.ISerializable Arabirim Tanımı

Serilize edilmek istenen nesnenin ISerializable arabirim gerçekleştirimini yapması durumunda .NET serilizasyon mimarisi ilgili nesnenin GetObjectData metodunu çağırmaktadır.

ISerializable arabirim gerçekleştiriminin yapıldığı durumda, de-serilizasyon amacıyla kullanılmak üzere sınıf için yapıcı metot tanımı da yapılmalıdır. Aşağıda örnek ISerializable arabirimi ile serilizasyon sürecinin özelleştirilmesini göstermektedir.

C#

[Serializable]

**public** **class** KullanıcıAyarları: ISerializable {

**public** **int** FormUstPozisyon;

**public** **int** FormSolPozisyon;

**public** **bool** BeniHatırla;

**public** KullanıcıAyarları() {

}

// Sadece istenilen alanlar de-serilize edilmektedir.

**protected** KullanıcıAyarları(SerializationInfo info,   
 StreamingContext context) {

FormUstPozisyon = info.GetInt32("Ust\_Pozisyon");

}

// Sadece istenilen alanlar serilize edilmektedir.

**public** **void** GetObjectData(SerializationInfo info,   
 StreamingContext context) {

info.AddValue("Ust\_Pozisyon", FormUstPozisyon);

}

}

VB.NET

<Serializable()> \_

**Public** **Class** KullanıcıAyarları

**Implements** ISerializable

**Public** FormUstPozisyon **As** **Integer**

**Public** FormSolPozisyon **As** **Integer**

**Public** BeniHatırla **As** **Boolean**

**Public** **Sub** **New**()

**End** **Sub**

' Sadece istenilen alanlar de-serilize edilmektedir.

**Protected** **Sub** **New**(**ByVal** info **As** SerializationInfo, \_  
 **ByVal** context **As** StreamingContext)

FormUstPozisyon = info.GetInt32("Ust\_Pozisyon")

**End** **Sub**

' Sadece istenilen alanlar serilize edilmektedir.

**Public** **Sub** GetObjectData(**ByVal** info **As** SerializationInfo, \_  
 **ByVal** context **As** StreamingContext)   
 **Implements** ISerializable.GetObjectData

info.AddValue("Ust\_Pozisyon", FormUstPozisyon)

**End** **Sub**

**End** **Class**

Kod 229, System.Runtime.Serialization.ISerializable Kullanım Örneği

ISerializable arabirimi ile serilizasyonun özelleştirilmesi desteklenmesine rağmen .NET 2.0 sürümünden itibaren bu işlevin öznitelikler kullanılarak sağlanması önerilmektedir.

Öznitelikler ve IDeserializationCallback Arabirimi

Aşağıda listelenen öznitelikler, System.Runtime.Serialization ad alanı içerisinde tanımlı olup serilizasyonun özelleştirilmesi amacıyla kullanılabilir.

| Öznitelik | Açıklama |
| --- | --- |
| OnSerializingAttribute | Serilize edilmeden önce çalıştırılması istenen metodu işaretlemek amacıyla kullanılmalıdır. |
| OnSerializedAttribute | Serilize edildikten sonra çalıştırılması istenen metodu işaretlemek amacıyla kullanılmalıdır. |
| OnDeserializingAttribute | De-serilize edilmeden önce çalıştırılması istenen metodu işaretlemek amacıyla kullanılmalıdır. |
| OnDeserializedAttribute | De-serilize edildikten sonra çalıştırılması istenen metodu işaretlemek amacıyla kullanılmalıdır. |

Tablo 92, Serilizasyon Özelleştirme Öznitelik Bilgileri

Serilizasyonun özelleştirilebilmesi için, ilgili nesne içerisinde değer döndürmeyen ve sadece StreamingContext tipinde bir parametre alan metodun tanımlanması ve amacına uygun öznitelik ile donatılması yeterlidir.

IDeserializationCallback arabirimi, de-serilizasyon işlemi tamamen tamamlandıktan sonra haberdar olmak amacıyla kullanılmalıdır.

C#

**public** **interface** IDeserializationCallback {

**void** OnDeserialization(**object** sender);

}

VB.NET

**Public** **Interface** IDeserializationCallback

**Sub** OnDeserialization(**ByVal** sender **As** **Object**)

**End** **Interface**

Kod 230, IDeserializationCallback Arabirim Tanımı

Öznitelikler ve IDeserializationCallback.OnDeserialization metodu;  
  
Serilizasyon için;   
1. OnSerializing  
2. OnSerialized  
  
De-Serilizasyon için;  
1. OnDeserializing  
2. IDeserializationCallback.OnDeserialization  
3. OnDeserialized  
  
sırasında çalıştırılmaktadır.

Dünyanın her yerinde herkesin yenileceği bir yer vardır. Kimilerini yenilgi yıkar, kimileriyse zaferle küçülür, bayağılaşırlar. Büyüklük, hem yenilgiyi, hem de zaferi kabullenebilen kişilerde yaşar.

John Steinbeck

Bölüm

15

İş Parçacıkları Kullanımı

İ

ş parçacıkları (ing. Threads), bir proses içerisindeki çalışma yollarına (ing. Thread of execution) verilen isimdir. İş parçacıkları kullanımı, bir programın farklı işlemleri eş zamanlı olarak yapabilmesini mümkün kılmaktadır.

Kitabın bu bölümünde, iş parçacıkları hakkında teorik düzeyde bilgiler ve .NET içerisinde iş parçacıklarının kullanım yöntemleri hakkında bilgiler aktarılacaktır.

Konuya başlamadan önce kitabın "Prosesler ve Uygulama Etki Alanları" bölümünün gözden geçirilmesi önerilmektedir.

Temel Kavramlar

İş parçacıkları kullanımı ile, uygulama tarafından sağlanan işlevlerin paralel olarak yürütülebilmesi, bunun doğal bir sonucu olarak da uygulamanın daha hızlı sonuç üretebilmesi sağlanmaktadır.

Şekil 39, Proses ve İş Parçacıkları

Yukarıdaki şekilde, Windows işletim sistemi, proses ve iş parçacıklarının genel yapısı gösterilmiştir.

Her proses içerisinde en az bir iş parçacığı bulunmaktadır ve bu iş parçacığı temel iş parçacığı (ing. Primary Thread) olarak tanımlanmaktadır. Temel iş parçacığı, yeni iş parçacıkları oluşturabilmekte, böylece proses içerisinde birden çok iş parçacığı paralel olarak çalışabilmektedir.

Proses içerisinde tek iş parçacığının bulunması durumunda uygulama tekli iş parçacık uygulaması (ing. Single-Threaded Application), birden çok iş parçacığının bulunması durumunda uygulama, çoklu iş parçacık uygulaması (ing. Multi-Threaded Application) olarak tanımlanmaktadır.

Paralel İşletim

Standart bir uygulama yazılımı, işletim sistemi ve donanım tarafından yapılan işlemler değerlendirildiğinde

* İşlemci tarafından yapılan matematiksel hesaplamalar.
* Sabit disk üzerinde bulunan dosyalara erişim, okuma ve yazma.
* Bilgisayar hafızasından veri okuma ve yazma (örneğin nesne alan değerleri veya yerel değişkenler).
* Kullanıcı girdisini bekleme.
* Uzak veritabanına, örneğin TCP (ing. Transmission Control Protocol) üzerinden, bağlanma ve SQL (ing. Structured Query Language) komutlarını iletme, dolayısıyla ağ üzerinden veri alış verişi sağlama.
* . . .

Gibi bir dizi işlemleri yapmaktadır. Bu işlemlerin tamamlanma süreleri değerlendirildiğinde, genellikle işlemci tarafından yapılan işlemler kısa sürelerde tamamlanabilmesine rağmen, diğer işlemler göreceli daha fazla zaman almaktadır. Bu durum paralel işletim kavramının temelini oluşturmaktadır.

Parallel Computing olarak ifade edilen ve birden çok bilgisayarın farklı işlemleri kendi kaynakları ile (işlemci, hafıza, disk, vb.) yapması ve sonuçların paralel olarak işlenmesi kavramı ile yukarıdaki kavram birbirine karıştırılmamalıdır.

Örneğin, uygulamanın sabit disk üzerinde bulunan bir dosyayı açabilmesi için, disk okuma kafasının fiziksel olarak gerekli pozisyona konumlanması, disk donanımının gerekli işlemleri yapması gerekmektedir. Bu işlemlerin 10ms (milisaniye) sürdüğü düşünüldüğünde, işlemci bu süre içerisinde faydalı herhangi bir işlem yapamamakta, sabit disk tarafından sonucun iletilmesini beklemektedir.

Çoklu görev işletim sistemleri (ing. Multi-Tasking Operating Systems), sistem üzerinde donanımsal olarak bir işlemcinin olması durumunda dahi, işlemciyi en verimli şekilde kullanabilmek amacıyla farklı algoritmalar içermekte, böylece birden çok uygulamayı paralel olarak çalıştırabilmektedir. Örneğin bir uygulamanın disk işlemini beklemesi durumunda işletim sistemi, işlemci kullanım hakkını bir başka uygulamaya tahsis etmektedir. Uygulama içerisinde birden çok iş parçacığının bulunması durumunda yine benzer işlem işletim sistemi tarafından yapılmaktadır.

Tekli İş Parçacık Uygulamaları

Her proses içerisinde en az bir iş parçacığı bulunmaktadır. Bir uygulama çalıştırıldığında yapılan işlem, temel iş parçacığının çalıştırılmasıdır.

Kitapta şu ana kadar verilen örnekler tekli iş parçacık uygulama örnekleridir.

**class** Program {

**static** **void** Main(**string**[] args) {

**string** s = Console.ReadLine();

Console.WriteLine(s);

}

}

Kod 231, Tekli İş Parçacık Uygulama Örneği

Yukarıdaki örnek C# kaynak kodu değerlendirildiğinde, proses içerisinde tek iş parçacığı bulunmaktadır. Temel iş parçacığı Main metodu ile başlamakta, bu metodun çalışması tamamlandığında temel iş parçacığının da çalışması tamamlanmaktadır.

Temel iş parçacığının çalışmasının tamamlanması prosesin, dolayısıyla uygulamanın da çalışmasının tamamlandığı anlamına gelmektedir.

Örnekte gösterilen basit konsol uygulaması kullanıcıdan klavye ile bilgi girişi yapmasını istemekte, bu işlemin ardından kullanıcı tarafından girilen bilgiyi konsol ekranına yazmakta ve çalışması tamamlanmaktadır. Kullanıcının bilgi girişi sırasında uygulamanın sadece beklediğine, bir başka işlem yapmadığına dikkat edilmelidir.

Çoklu İş Parçacık Uygulamaları

Çoklu iş parçacık uygulamaları, içerisinde birden çok iş parçacığını bulunduran uygulamalardır. Bu uygulamaların paralel olarak farklı işlemleri yapabilmesi mümkündür.

Çoklu iş parçacık uygulamalarına örnek olarak ilişkisel veritabanı yönetim sistemleri verilebilir. İlişkisel veritabanı yönetim sistemleri, eş zamanlı olarak birçok kullanıcıya hizmet verebilmekte, bir kullanıcının işlemi devam ederken bir başka kullanıcı veritabanı yönetim sistemine bağlanıp eş zamanlı olarak SQL komutlarını çalıştırabilmektedir.

Uygulama ve iş ihtiyaçları değerlendirildiğinde, her uygulamanın birden çok iş parçacığı bulundurması gerekmeyebilir. Örneğin işlemlerin paralel olarak yürütülmesinin mümkün olmadığı durumlarda çoklu iş parçacık kullanımı avantaj sağlayamayabilir. Bir uygulama kullanıcıdan bilgi girmesini beklerken aynı zamanda sabit diskteki bir dosyayı okuyabilir veya eş zamanlı olarak yazıcı çıktısı üretebilir. Bu işlemlerin paralel yapılamaması, kullanıcı girdisi tamamlanmadan dosya okumanın başlayamaması, dosya okuma işlemi tamamlanmadan yazıcı çıktı üretilememesi durumunda eş zamanlı işletimin sağlayabileceği herhangi bir kazanım bulunmamaktadır.

Çoklu iş parçacık uygulamalarının tasarımı ve geliştirilmesi, tekli iş parçacık uygulamalarına göre genellikle daha karmaşıktır. Özellikle iş parçacıklarının birbirleri ile haberleşmesi ve ortak kaynak (ing. Shared Resource) kullanımı gibi konuların iyi değerlendirilmemesi sonucu, bazen düzgün sonuç alınması bazen alınamaması, uygulamanın tepki vermemesi gibi olumsuz durumlar ortaya çıkabilir, hatanın nereden kaynaklandığının anlaşılması oldukça zor olabilir.

İş Parçacık ve Kaynak Senkronizasyonu

İş ihtiyaçlarının değerlendirilmesi sonucu, uygulama içerisinde birden çok iş parçacığının bulunabilmesi ve iş parçacıklarının eş zamanlı olarak çalışabilmesi mümkün olmasına rağmen, iş parçacıkları paylaşılan bir kaynağa erişmek isteyebilirler.

Paylaşılan kaynak ile ifade edilmek istenen kavram, birden çok iş parçacığının erişebileceği ve tek olan kaynaktır. Örneğin yazıcı paylaşılan bir kaynaktır. Eş zamanlı çalışan iş parçacıklarının yazıcıya erişiminin senkronize edilmemesi, yazıcı bir iş parçacığı tarafından kullanılırken diğer iş parçacıklarının erişiminin düzenlenmemesi önemli senkronizasyon problemlerini ortaya çıkarabilir. Aynı durum bilgisayar hafızası için de geçerlidir.

Benzer şekilde, bir iş parçacığının çalışması tamamlanmadan bir başka iş parçacığının çalışmaya başlamaması gerekebilir. Bu durumda iki iş parçacığının senkronize edilmesi gerekebilir.

İş Parçacık Güvenli Kavramı

İş parçacık güvenli (ing. Thread Safe), bir kod parçasına eş zamanlı olarak birden çok iş parçacığının erişmesi durumunda, ilgili kod parçasının ortak kaynaklar için gerekli senkronizasyonu sağladığını belirten kavramdır.

Örneğin aynı kod parçasını eş zamanlı olarak çalıştıran iki iş parçacığı değerlendirildiğinde, ilk çalışan iş parçacığı ortak kullanılan herhangi bir hafıza alanına değer atadıktan hemen sonra, bir başka iş parçacığı ilgili değeri değiştirebilir. Bu durum, ilk iş parçacığının hafıza alanını daha sonra okuduğunda kendi yazdığından farklı bir değer bulmasına, dolayısıyla potansiyel problemlere yol açabilir.

.NET sınıf kütüphanesi içerisinde, iş parçacık güvenli ve güvenli olmayan sınıf ve üyeler bulunmaktadır. Her sınıf ve üyeye ait dokümantasyonda, ilgili kaynak kodun iş parçacık güvenli olup olmadığı bilgisi bulunmaktadır.

.NET İş Parçacık Mimarisi

.NET iş parçacık mimarisinde, bir önceki konuda teorik düzeyde anlatılan iş parçacığı kavramı System.Threading.Thread sınıfı ile ifade edilmektedir.

System.Threading ad alanı içerisinde çoklu iş parçacık uygulamalarında kullanılabilecek tip tanımları bulunmaktadır.

İş Parçacık Sınıfı Bilgileri

Aşağıda Thread sınıfı üye bilgileri ve açıklamaları listelenmiştir.

| Üye Adı | Açıklama |
| --- | --- |
| CurrentContext | İş parçacığının içerisinde çalıştığı bağlam (ing. Context) nesnesini döndüren statik özelliktir. |
| CurrentCulture | İş parçacığına ait kültür bilgisini (ing. CultureInfo) döndüren statik özelliktir. Kültür bilgisi hakkında detaylı bilgi kitabın .NET Assembly Bilgileri başlıklı konusunda verilmiştir. |
| CurrentPrincipal | İş parçacığının hangi kullanıcı ve yetkiler (rol bazlı yetkilendirme için) kapsamında çalıştığını döndüren statik özelliktir. |
| CurrentThread | Aktif iş parçacık nesnesini döndüren statik özelliktir. |
| CurrentUICulture | Assembly kaynak yöneticisi (ing. ResourceManager) tarafından kullanılan kültür bilgisini döndüren özelliktir. Kaynak yöneticisi hakkında detaylı bilgi kitabın .NET Assembly Bilgileri başlıklı konusunda verilmiştir. |
| IsAlive | İş parçacığının aktif olup olmadığını döndüren özelliktir. |
| IsBackground | İş parçacığının arka planda çalışan iş parçacığı olup olmadığını döndüren özelliktir. |
| ManagedThreadId | İş parçacığını tekil olarak ifade eden tamsayı değeri döndüren özelliktir. |
| Name | İş parçacığına verilen karakter dizi değeri döndüren özelliktir. |
| Priority | İş parçacığının öncelik bilgisini döndüren özelliktir. |
| ThreadState | İş parçacığının durumunu döndüren özelliktir. |
| Abort | İş parçacığının çalışmasını durdurmak amacıyla iş parçacığına dur sinyalini ileten metottur. |
| BeginCriticalSection | Metot çağrımından sonra çalıştırılacak kodun kritik olduğunu, bir başka anlatımla kodun çalışmasının tamamlanamaması durumunda uygulama etki alanının kararsızlaşabileceğini belirtmek için kullanılan statik metottur. |
| EndCriticalSection | Kritik kod bloğunun tamamlandığını belirtmek için çağrılan statik metottur. |
| GetDomain | İş parçacığının içerisinde çalıştığı uygulama etki alanı nesnesini döndüren metottur. Uygulama etki alanları hakkında detaylı bilgi kitabın Prosesler ve Uygulama Etki Alanları başlıklı konusunda verilmiştir. |
| Interrupt | WaitSleepJoin durumunda olan iş parçacığına müdahale eder ve ThreadInterruptedException istisnası fırlatır. |
| Join | Çağıran iş parçacığını, çağrılan iş parçacığının çalışması tamamlanana kadar bloke eden metottur. |
| ResetAbort | Abort metodu çağrısı ile iletilen dur sinyalini iptal etmek için kullanılması gereken metottur. |
| Sleep | Parametre olarak iletilen milisaniye değeri kadar iş parçacığı çalışmasını durduran metottur. |
| Start | İş parçacığına çalış sinyalini ileten metottur. |
| VolatileRead | İstenilen alan değerini okumak için kullanılan statik metottur. |
| VolatileWrite | İstenilen alana bilgisayarda bulunan tüm işlemciler tarafından eş zamanlı olarak görülebilecek şekilde değer yazmayı sağlayan statik metottur. |

Tablo 93, System.Threading.Thread Sınıfı Üye Bilgileri

Yardımcı Tip Tanımları

System.Threading.ThreadPriority sayılabilir tipi iş parçacığının öncelik bilgisini ifade etmek için kullanılan tiptir.

| C# | VB.NET |
| --- | --- |
| **public** **enum** ThreadPriority {  // En düşük öncelik  Lowest,  // Normal altı öncelik  BelowNormal,  // Normal öncelik  Normal,  // Normal üstü öncelik  AboveNormal,  // En yüksek öncelik  Highest  } | **Public** **Enum** ThreadPriority  ' En düşük öncelik  Lowest  ' Normal altı öncelik  BelowNormal  ' Normal öncelik  Normal  ' Normal üstü öncelik  AboveNormal  ' En yüksek öncelik  Highest  **End** **Enum** |

Kod 232, System.Threading.ThreadPriority Tip Tanımı

İş parçacığının önceliği, işlemci gücünün ne düzeyde ilgili iş parçacığına tahsis edileceğini belirtmektedir.

System.Threading.ThreadState sayılabilir tipi iş parçacığının durumunu ifade etmek için kullanılan tiptir.

| C# | VB.NET |
| --- | --- |
| **public** **enum** ThreadState {  Aborted = 0x100,  AbortRequested = 0x80,  Background = 4,  . . .  } | **Public** **Enum** ThreadState  Aborted = &H100  AbortRequested = &H80  Background = 4  . . .  **End** **Enum** |

Kod 233, System.Threading.ThreadState Tip Tanımı

Aşağıda pratikte kullanılan tip üyelerinin anlamları listelenmiştir.

| Üye Adı | Açıklama |
| --- | --- |
| Aborted | İş parçacığının Thread.Abort metodu çağrısı ile dur sinyalini aldığını ancak iş parçacık durum değerinin henüz Stopped olmadığını ifade eder. |
| AbortRequested | İş parçacığına Thread.Abort metodu çağrısı ile dur sinyalinin iletildiğini ifade eder. |
| Background | İş parçacığının arka plan iş parçacığı olduğunu ifade eder. Değer, Thread.IsBackground özelliği ile kontrol edilmektedir. |
| Running | İş parçacığının çalıştığını ifade eder. |
| Stopped | İş parçacığının çalışmasının durduğunu ifade eder. |
| UnStarted | İş parçacığının henüz çalışmaya başlamadığını ifade eder. |
| WaitSleepJoin | İş parçacığının çalışmasının bloke edildiğini ifade eder. |

Tablo 94, System.Threading.ThreadState Tipi Üye Bilgileri

Ön Plan ve Arka Plan İş Parçacıkları

Thread.IsBackground özelliği iş parçacığının ön plan veya arka plan iş parçacığı niteliğini kontrol eden özelliktir. Bu özelliğe True değerinin atanması iş paçacığının arka plan, False değerinin atanması ön plan olmasını sağlamaktadır.

Her iş parçacığı ön plan veya arka plan niteliğine sahip olmalıdır. Bu nitelik ile, prosesin sonlanması durumunda CLR davranışının nasıl olacağı belirlenmektedir.

Bir prosesin sonlanabilmesi için, çalışan ön plan iş parçacıklarının çalışmasının tamamlanması gerekmektedir. Arka plan çalışan iş parçacıkları, bundan farklı olarak, proses sonlanırken CLR tarafından durdurulmaktadır.

İş Parçacık Kullanım Bilgileri

Bu bölümde;

* İş parçacıkları hakkında bilgi alma
* İş parçacık nesneleri oluşturma
* İş parçacıklarının başlatılması ve durdurulması
* Kritik kod bölümleri

Konularında bilgiler verilecektir.

İş Parçacıkları Genel Özellikleri

Bilindiği gibi iş parçacıkları program içerisindeki çalışma yollarını ifade etmektedir. Her programda en az bir adet olmak üzere istenilen sayıda iş parçacığı bulunabilmektedir.

Thread sınıfı ile ifade edilen iş parçacığı, çeşitli bilgilere erişim için üyeler içermektedir.

C#

**static** **void** ThreadBilgisiGoster(Thread t) {

Console.WriteLine("İş Parçacık Id: {0}", t.ManagedThreadId);

Console.WriteLine("İş Parçacık Durumu: {0}", t.ThreadState);

Console.WriteLine("İş Parçacık Önceliği: {0}", t.Priority);

Console.WriteLine("İş Parçacık Arka Plan ?: {0}", t.IsBackground);

Console.WriteLine("Aktif İş Parçacık Kullanıcı: {0}",   
 Thread.CurrentPrincipal.Identity.Name);

Console.WriteLine("Aktif İş Parçacık Kimlik Doğrulama Yöntemi: {0}",   
 Thread.CurrentPrincipal.Identity.AuthenticationType);

}

**static** **void** Main(**string**[] args) {

Thread t = Thread.CurrentThread;

ThreadBilgisiGoster(t);

}

VB.NET

**Shared** **Sub** ThreadBilgisiGoster(**ByVal** t **As** Thread)

Console.WriteLine("İş Parçacık Id: {0}", t.ManagedThreadId)

Console.WriteLine("İş Parçacık Durumu: {0}", t.ThreadState)

Console.WriteLine("İş Parçacık Önceliği: {0}", t.Priority)

Console.WriteLine("İş Parçacık Arka Plan ?: {0}", t.IsBackground)

Console.WriteLine("Aktif İş Parçacık Kullanıcı: {0}", \_

Thread.CurrentPrincipal.Identity.Name)

Console.WriteLine("Aktif İş Parçacık Kimlik Doğrulama Yöntemi: {0}", \_

Thread.CurrentPrincipal.Identity.AuthenticationType)

**End** **Sub**

**Shared** **Sub** Main(**ByVal** args() **As** **String**)

**Dim** t **As** Thread = Thread.CurrentThread

ThreadBilgisiGoster(t)

**End** **Sub**

Kod 234, İş Parçacığı Hakkında Bilgi Alma

Yukarıdaki kod örneğinde Thread.CurrentThread statik metodu ile aktif iş parçacık nesnesi elde edilmektedir. ThreadBilgisiGoster metoduna parametre olarak iletilen iş parçacık nesnesi hakkında çeşitli bilgiler ekrana yazdırılmaktadır.

Yukarıdaki kod parçasının basit bir konsol uygulaması olduğuna dikkat edilmelidir. Thread.CurrentThread özelliği ile elde edilen iş parçacığı, tekli iş parçacık uygulaması için, temel iş parçacık nesnesidir. Temel iş parçacık nesnesi CLR tarafından program çalıştırıldığında oluşturulmaktadır.

Her iş parçacığı kendi içerisinde bir bütündür. Bu bütünlük içerisinde, iş parçacığının içerisinde çalıştığı bağlam, kullanıcı kimlik doğrulama (ing. Authentication) ve yetki (ing. Authorization) bilgisi bulunmaktadır. Örnek kod parçası içerisinde, iş parçacığının kimlik doğrulama bilgilerine erişim gösterilmiştir.

İş Parçacık Kullanım Detayları

Çoklu iş parçacık uygulamalarında, CLR tarafından program çalışmaya başladıktan sonra otomatik olarak oluşturulan temel iş parçacığı dışında iş parçacıkları da bulunabilmektedir.

Bu konu başlığında programcı tarafından yeni iş parçacık nesnelerinin oluşturulması ve kullanımı konusunda temel bilgiler verilecektir.

Temel Bilgiler

Bilindiği gibi temel iş parçacığının çalışmasının tamamlanması programın sonlanması anlamına gelmektedir.

| C# | VB.NET |
| --- | --- |
| **static** **void** MesajYaz() {  Console.WriteLine("Test");  }  **static** **void** Main(**string**[] args) {  MesajYaz();  } | **Shared** **Sub** MesajYaz()  Console.WriteLine("Test")  **End** **Sub**  **Shared** **Sub** Main(**ByVal** args()   **As** **String**)  MesajYaz()  **End** **Sub** |

Kod 235, Tekli İş Parçacık Uygulaması Metodu

Örnek kod parçasında, program çalıştırıldığında MesajYaz isimli metot çağrılmakta ve program sonlanmaktadır.

Main isimli metot, program çalıştırıldığında ilk çağrılan metottur. Arka planda CLR tarafından yapılan işlem, temel iş parçacık nesnesinin oluşturulması ve temel iş parçacık nesnesinin Main metodunu çağırmasıdır. Main metodunun çalışmasının tamamlanması, temel iş parçacığının çalışmasının tamamlanmasını, dolayısıyla da programın sonlanmasını sağlamaktadır.

.NET içerisinde çalıştırılan her kod, bir iş parçacığı tarafından çalıştırılmaktadır. Örnekte de dikkat edilmesi gereken en önemli konu, MesajYaz isimli metodun temel iş parçacığı tarafından çalıştırıldığıdır.

İş Parçacıkları Oluşturma ve Çalıştırma

Programcı tarafından yeni iş parçacıkları oluşturulması, yeni iş parçacık nesnelerinin oluşturulması anlamını taşımaktadır.

Pratikte en sık kullanılan Thread sınıfı yapıcı metot tanımları aşağıda gösterilmiştir.

C#

**public** Thread(ThreadStart start);

**public** Thread(ParameterizedThreadStart start);

VB.NET

**Public** **Sub** **New**(**ByVal** start **As** ThreadStart)

**Public** **Sub** **New**(**ByVal** start **As** ParameterizedThreadStart)

Kod 236, System.Threading.Thread Sınıfı Yapıcı Metot Tanımları

Yapıcı metotlara parametre olarak iletilen ThreadStart ve ParameterizedThreadStart delegeleri, iş parçacığı başladığında çağrılması istenen metodu ifade etmektedir.

C#

**public** **delegate** **void** ThreadStart()

**public** **delegate** **void** ParameterizedThreadStart(**object** obj);

VB.NET

**Public** **Delegate** **Sub** ThreadStart()

**Public** **Delegate** **Sub** ParameterizedThreadStart(**ByVal** obj **As** **Object**)

Kod 237, ThreadStart ve ParameterizedThreadStart Delegeleri

Delege tanımlarından, ThreadStart delegesinin herhangi bir parametre almadığı, ParameterizedThreadStart delegesinin bir adet Object tipinde parametre aldığı görülmektedir.

C#

**static** **void** MesajYaz() {

Console.Write("İş parçacık No:");

Console.WriteLine(Thread.CurrentThread.ManagedThreadId);

}

**static** **void** Main(**string**[] args) {

MesajYaz();

ThreadStart ts = **new** ThreadStart(MesajYaz);

Thread t = **new** Thread(ts);

// veya Thread t = new Thread(MesajYaz)

t.Start();

}

VB.NET

**Shared** **Sub** MesajYaz()

Console.Write("İş parçacık No:")

Console.WriteLine(Thread.CurrentThread.ManagedThreadId)

**End** **Sub**

**Shared** **Sub** Main(**ByVal** args() **As** **String**)

MesajYaz()

**Dim** ts **As** ThreadStart = **New** ThreadStart(**AddressOf** MesajYaz)

**Dim** t **As** Thread = **New** Thread(ts)

' veya Dim t As Thread = New Thread(addressof MesajYaz)

t.Start()

**End** **Sub**

Kod 238, İş Parçacık Nesnesi Oluşturulması ve Basit Kullanımı

Bir iş parçacık nesnesinin oluşturulması, çalıştırılmak için hazırlanması ve çalışmaya başlamasının ayrı adımlar olduğuna dikkat edilmelidir.

Örnekte, iş parçacık nesnesinin oluşturulması ardından Thread.Start metodu çağrılmaktadır. Bu metodun çağrılması ile iş parçacık nesnesi çalıştırılmak için hazırlanmaktadır. İş parçacığının ne zaman gerçekten çalışmaya başlayacağına CLR ve işletim sistemi tarafından kadar verilmektedir.

Kod çıktısı aşağıdaki gibi olmaktadır. İş parçacık numaralarının değişebileceğine dikkat edilmelidir.

İş parçacık No:10

İş parçacık No:11

İlk çalışmaya başlayan iş parçacığı temel iş parçacığıdır. Daha sonra, programcı tarafından oluşturulan iş parçacığı çalışmaya başlamaktadır. Böylece aynı metot farklı iş parçacıkları tarafından çalıştırılmaktadır.

ParameterizedThreadStart delegesi ile hedeflenen amaç, iş parçacığı tarafından çalıştırılan metoda istenilen bilginin iletiminin sağlanmasıdır.

C#

**static** **void** MesajYaz(**object** mesaj) {

Console.Write("İş parçacık No:");

Console.WriteLine(Thread.CurrentThread.ManagedThreadId);

Console.WriteLine(mesaj);

}

**static** **void** Main(**string**[] args) {

MesajYaz("Temel iş parçacığı");

ParameterizedThreadStart ts = **new** ParameterizedThreadStart(MesajYaz);

Thread t = **new** Thread(ts);

t.Start("Yeni iş parçacığı");

}

VB.NET

**Shared** **Sub** MesajYaz(**ByVal** mesaj **As** **Object**)

Console.Write("İş parçacık No:")

Console.WriteLine(Thread.CurrentThread.ManagedThreadId)

Console.WriteLine(mesaj)

**End** **Sub**

**Shared** **Sub** Main(**ByVal** args() **As** **String**)

MesajYaz("Temel iş parçacığı")

**Dim** ts **As** ParameterizedThreadStart

ts= **New** ParameterizedThreadStart(**AddressOf** MesajYaz)

**Dim** t **As** Thread = **New** Thread(ts)

t.Start("Yeni iş parçacığı")

**End** **Sub**

Kod 239, System.Threading.ParameterizedThreadStart Kullanımı

İletilen parametre System.Object tipindedir. Böylece her tip nesnenin metoda iletimi mümkün olmaktadır.

Kod çıktısı aşağıda gösterilmiştir.

İş parçacık No:11

Temel iş parçacığı

İş parçacık No:12

Yeni iş parçacığı

İş parçacığının gerçekten çalışmaya başladığı Thread.IsAlive özelliğinin true değerini döndürmesi ile anlaşılabilir. Aşağıdaki kod örneğinde, iş parçacığının Start metodu çağrılmakta ve iş parçacığı gerçekten çalışana kadar beklenmektedir.  
  
C#  
Thread t = new Thread(ts);  
t.Start();  
while (!t.IsAlive);  
  
VB.NET  
Dim t As Thread = new Thread(ts)  
t.Start()  
While Not t.IsAlive  
End While

Çalışmayı Bekletme

Thread.Sleep metodu iş parçacığının çalışmasını istenilen süre kadar bekletmek (duraklatmak) için kullanılmaktadır.

Aşağıdaki örnekte 3 adet iş parçacık nesnesi oluşturulmakta ve iş parçacıklarının çalışması 100ms kadar bekletilmektedir. Aynı zamanda bazı iş parçacıklarının öncelik değeri daha yüksek olarak belirlenmektedir.

C#

**static** **void** MesajYaz() {

**for** (**int** i = 1; i < 3; i++) {

Console.WriteLine("İş parçacık No: {0} bekliyor ...",   
 Thread.CurrentThread.ManagedThreadId);

Thread.Sleep(100);

}

}

**static** **void** Main(**string**[] args) {

ThreadStart ts = **new** ThreadStart(MesajYaz);

Thread t;

**for** (**int** i = 1; i < 4; i++) {

t = **new** Thread(ts);

**if** (i % 2 == 0)

t.Priority = ThreadPriority.AboveNormal;

t.Start();

}

}

VB.NET

**Shared** **Sub** MesajYaz()

**For** i **As** **Integer** = 1 **To** 2

Console.WriteLine("İş parçacık No: {0} bekliyor ...", \_  
 Thread.CurrentThread.ManagedThreadId)

Thread.Sleep(100)

**Next**

**End** **Sub**

**Shared** **Sub** Main(**ByVal** args() **As** **String**)

**Dim** ts **As** ThreadStart = **New** ThreadStart(**AddressOf** MesajYaz)

**Dim** t **As** Thread

**For** i **As** **Integer** = 1 **To** 3

t = **New** Thread(ts)

**If** i **Mod** 2 = 0 **Then**

t.Priority = ThreadPriority.AboveNormal

**End** **If**

t.Start()

**Next**

**End** **Sub**

Kod 240, İş Parçacıklarının Bekletilmesi

Kod parçası çıktısı aşağıdaki gibi olmaktadır.

İş parçacık No: 11 bekliyor ...

İş parçacık No: 12 bekliyor ...

İş parçacık No: 13 bekliyor ...

**İş parçacık No: 12 bekliyor ...**

İş parçacık No: 11 bekliyor ...

İş parçacık No: 13 bekliyor ...

12 numaralı iş parçacığının diğerlerine göre daha öncelikli çalıştığına dikkat edilmelidir.

Çalışmanın Tamamlanmasını Bekleme

Thread.Join metodu iş parçacığının çalışmasını beklemek için kullanılmaktadır. Bu metodu çağıran iş parçacığı, ilgili iş parçacık nesnesinin çalışması tamamlanana kadar bloke olmaktadır.

Aşağıdaki örnekte Thread.Join metodu kullanımı gösterilmiştir.

C#

**static** **void** MesajYaz() {

**for** (**int** i = 1; i < 3; i++) {

Console.WriteLine("İş parçacık No: {0}",   
 Thread.CurrentThread.ManagedThreadId);

Thread.Sleep(5000);

}

}

**static** **void** Main(**string**[] args) {

Thread[] list = **new** Thread[3];

ThreadStart ts = **new** ThreadStart(MesajYaz);

**for** (**int** i = 0; i < 3; i++) {

Thread t = **new** Thread(ts);

list[i] = t;

t.Start();

}

Console.WriteLine("İş parçacıklarının tamamlanması bekleniyor ...");

**for** (**int** i = 0; i < 3; i++) {

((Thread)list[i]).Join();

}

Console.WriteLine("İş parçacıklarının çalışması tamamlandı ...");

}

VB.NET

**Shared** **Sub** MesajYaz()

**For** i **As Integer** = 1 **To** 2

Console.WriteLine("İş parçacık No: {0}", \_  
 Thread.CurrentThread.ManagedThreadId)

Thread.Sleep(5000)

**Next**

**End** **Sub**

**Shared** **Sub** Main(**ByVal** args() **As** **String**)

**Dim** list(2) **As** Thread

**Dim** ts **As** ThreadStart = **New** ThreadStart(**AddressOf** MesajYaz)

**For** i **As** **Integer** = 0 **To** 2

**Dim** t **As** Thread = **New** Thread(ts)

list(i) = t

t.Start()

**Next**

Console.WriteLine("İş parçacıklarının tamamlanması bekleniyor ...")

**For** j **As** **Integer** = 0 **To** 2

**CType**(list(j), Thread).Join()

**Next**

Console.WriteLine("İş parçacıklarının çalışması tamamlandı ...")

**End** **Sub**

Kod 241, System.Threading.Thread.Join Kullanım Örneği

Örnekte, içerisinde 3 adet iş parçacık nesnesi bulunan dizi bulunmaktadır. Program sonlanmadan önce Thread.Join metodu ile oluşturulan iş parçacıklarının tamamlanması beklenmektedir.

İş Parçacığının Çalışmasının İptal Edilmesi

Thread.Abort metodu iş parçacığının çalışmasını iptal etmek için kullanılmaktadır.

Thread.Abort metodu ile arka planda yapılan işlem ThreadAbortException istisnasının fırlatılmasıdır. İstisnanın fırlatılması ile iş parçacığının kod çalıştırılması durdurulmaktadır.

İş parçacığının çalışmasının Thread.Abort metodu çağrısı ile iptal edilmesi ile iş parçacığı durumu (Thread.ThreadState) ThreadState.Aborted değerini almaktadır.

Thread.Start metodunda olduğu gibi, Thread.Abort metodu çağrısı da anında işlem yapmamaktadır. Thread.Abort metodu çağrısı ile iş parçacığına iptal sinyali iletilmektedir. Sinyalin iletilmesi ile iş parçacık durumu ThreadState.AbortRequested değerini almaktadır.

Aşağıdaki örnekte, MesajYaz isimli metot içerisinde iş parçacığının çalışmasının iptal edilip edilmediği kontrol edilmektedir.

C#

**static** **void** MesajYaz() {

**try** {

**for** (**int** i = 1; i < 3; i++) {

Console.WriteLine("İş parçacık No: {0}",  
 Thread.CurrentThread.ManagedThreadId);

Thread.Sleep(2000);

}

}

**catch** (ThreadAbortException) {

Console.WriteLine("İşlem iptal edildi ...");

}

}

**static** **void** Main(**string**[] args) {

Thread[] list = **new** Thread[3];

ThreadStart ts = **new** ThreadStart(MesajYaz);

**for** (**int** i = 0; i < 3; i++) {

Thread t = **new** Thread(ts);

list[i] = t;

t.Start();

}

// Thread.Abort metodu çağrısı ile devam eden çalışma iptal  
 // edilmektedir.

**for** (**int** i = 0; i < 3; i++) {

((Thread)list[i]).Abort();

}

}

VB.NET

**Shared** **Sub** MesajYaz()

**Try**

**For** i **As** **Integer** = 1 **To** 2

Console.WriteLine("İş parçacık No: {0}", \_  
 Thread.CurrentThread.ManagedThreadId)

Thread.Sleep(2000)

**Next**

**Catch** e **As** ThreadAbortException

Console.WriteLine("İşlem iptal edildi ...")

**End** **Try**

**End** **Sub**

**Shared** **Sub** Main(**ByVal** args() **As** **String**)

**Dim** list(2) **As** Thread

**Dim** ts **As** ThreadStart = **New** ThreadStart(**AddressOf** MesajYaz)

**For** i **As** **Integer** = 0 **To** 2

**Dim** t **As** Thread = **New** Thread(ts)

list(i) = t

t.Start()

**Next**

' Thread.Abort metodu çağrısı ile devam eden çalışma iptal  
 ' edilmektedir.

**For** j **As** **Integer** = 0 **To** 2

**CType**(list(j), Thread).Abort()

**Next**

**End** **Sub**

Kod 242, System.Threading.Thread.Abort Kullanım Örneği

ThreadAbortException, CLR tarafından diğer istisna sınıflarından farklı olarak ele alınmaktadır. Yakalanmayan bir istisna programın kapatılmasına sebep olurken, ThreadAbortException istisnasının yakalanmaması programın kapatılmasına sebep olmamaktadır.

Kritik Kod Blokları

Thread.Abort metodunun çağrılması ile iş parçacığının çalışmasının iptal edilmesi için sinyal iletilmekte ve ThreadAbortException istisnası fırlatılmaktadır.

Thread.Abort metodu çağrıldığında, ilgili iş parçacık nesnesi herhangi bir kod bloğunu çalıştırıyor olabilir. ThreadAbortException istisnasının fırlatılması ile o anda çalıştırılan kodun çalıştırılması durdurulmaktadır.

Bazı durumlarda, iş parçacığının çalışması iptal edilse bile belirli bir kod bloğunun çalışmasının tamamlanması gerekebilir. Bir başka anlatımla, içerisinde bir veya daha fazla ifade (ing. Statement) içeren kod bloğunun çalışması tamamlanmadan ThreadAbortException istisnasının fırlatılmaması gerekebilir.

C#

**static** **void** IslemYap() {

// Diğer İşlemler

[1] FileStream f = **new** FileStream(@"c:\test.dat", FileMode.Create);

[2] Console.WriteLine("Dosya oluşturuldu ...");

[3] BinaryWriter w = **new** BinaryWriter(f);

[4] w.Write("Örnek değer ...");

[5] Console.WriteLine("Dosyaya yazıldı ...");

[6] f.Close();

[7] Console.WriteLine("Dosya kapatıldı ...");

// Diğer İşlemler

}

**static** **void** Main(**string**[] args) {

ThreadStart ts = **new** ThreadStart(IslemYap);

Thread t = **new** Thread(ts);

t.Start();

t.Abort();

}

VB.NET

**Shared** **Sub** IslemYap()

' Diğer İşlemler

[1] **Dim** f **As** FileStream = **New** FileStream("c:\test.dat", FileMode.Create)

[2] Console.WriteLine("Dosya oluşturuldu ...")

[3] **Dim** w **As** BinaryWriter = **New** BinaryWriter(f)

[4] w.Write("Örnek değer ...")

[5] Console.WriteLine("Dosyaya yazıldı ...")

[6] f.Close()

[7] Console.WriteLine("Dosya kapatıldı ...")

' Diğer İşlemler

**End** **Sub**

**Shared** **Sub** Main(**ByVal** args() **As** **String**)

**Dim** ts **As** ThreadStart = **New** ThreadStart(**AddressOf** IslemYap)

**Dim** t **As** Thread = **New** Thread(ts)

t.Start()

t.Abort()

**End** **Sub**

Kod 243, Örnek İş Parçacık Bloğu

Yukarıdaki örnekte Thread.Start metodu ile iş parçacığının çalışması başlamakta ve hemen ardından Thread.Abort metodu çağrılmaktadır. İş parçacığının başlaması ile çalışan IslemYap metodu içerisindeki kodların çalışması devam ederken, Thread.Abort metodu temel iş parçacığı tarafından çağrılmaktadır.

Bu durumda IslemYap metodu içerisinde hangi satır çalıştırılır iken ThreadAbortException istisnasının fırlatılacağı belirli değildir. Örneğin dosya açılmışken [1] veya konsola mesaj yazılırken [7] bu istisna fırlatılabilir.

Kritik kod blokları, iş parçacığının Thread.Abort çağrısı ile iptal edilmesi durumunda dahi çalıştırılması istenen kod bloğunu ifade etmek için kullanılmaktadır.

Kritik kod bloğu başlangıcı Thread.BeginCriticalRegion, bitimi Thread.EndCriticalRegion statik metotları ile belirtilmektedir.

Yukarıda verilen örnek, kritik kod blokları kullanılarak aşağıdaki gibi düzenlenebilir.

C#

**static** **void** IslemYap() {

// Diğer İşlemler

Thread.BeginCriticalRegion();

[1] FileStream f = **new** FileStream(@"c:\test.dat", FileMode.Create);

[2] Console.WriteLine("Dosya oluşturuldu ...");

[3] BinaryWriter w = **new** BinaryWriter(f);

[4] w.Write("Örnek değer ...");

[5] Console.WriteLine("Dosyaya yazıldı ...");

[6] f.Close();

[7] Console.WriteLine("Dosya kapatıldı ...");

Thread.EndCriticalRegion();

// Diğer İşlemler

}

VB.NET

**Shared** **Sub** IslemYap()

' Diğer İşlemler

Thread.BeginCriticalRegion()

[1] **Dim** f **As** FileStream = **New** FileStream("c:\test.dat", FileMode.Create)

[2] Console.WriteLine("Dosya oluşturuldu ...")

[3] **Dim** w **As** BinaryWriter = **New** BinaryWriter(f)

[4] w.Write("Örnek değer ...")

[5] Console.WriteLine("Dosyaya yazıldı ...")

[6] f.Close()

[7] Console.WriteLine("Dosya kapatıldı ...")

Thread.EndCriticalRegion()

' Diğer İşlemler

**End** **Sub**

Kod 244, Kritik Kod Blokları Kullanım Örneği

Kritik kod blokları kullanımı ile, örneğin [3] numaralı satır iş parçacığı tarafından çalıştırılırken iptal sinyali alınsa dahi [7] numaralı satırın çalışması tamamlanmadan ThreadAbortException istisnası fırlatılmayacaktır.

Kritik kod blokları kullanımı ile hedeflenen, bir bütün olarak çalışması gereken kod bloğunu belirtmek ve kritik kod bloğunun tamamlanamaması durumunda uygulama etki alanının kararsızlaşabileceğini ifade etmektir.

İş Parçacık Senkronizasyonu ve Kaynak Paylaşımı

Farklı iş parçacıklarının ortak kaynaklara erişimi ve erişimin senkronize edilmesi çoklu iş parçacık uygulamalarında en çok dikkat edilmesi gereken konulardan birisidir.

İş parçacık senkronizasyonu ve farklı iş parçacıklarının ortak kaynaklara erişimlerinin düzenlenmesi geniş ve detaylı bir konudur. Kitabın bu bölümünde pratikte en çok kullanılan yöntemler konusunda olabildiğince detaylı bilgi sunulacaktır.

Ortak Kaynak Kullanım Temel Bilgileri

İş parçacık senkronizasyonu ve ortak kaynak kullanımı konusunda en güzel örnek veritabanı yönetim sistemleridir. Farklı kullanıcılara eş zamanlı olarak hizmet verebilen veritabanı yönetim sistemi yazılımları çoklu iş parçacık uygulamalarıdır. Verinin tek ancak veriyi kullanan iş parçacıklarının (örneğin kullanıcı oturumlarının) istenilen sayıda olabildiği bu yapıda, farklı iş parçacıklarının veriye erişimi, veriyi düzenlemeleri belirli kurallar ile yönetilmektedir. Örneğin herhangi bir kullanıcının başlattığı işlem grubu (ing. Transaction) içerisinde bir kaydın değerinin değiştirilmesi, aynı kayda diğer kullanıcıların erişimini bloke edebilmektedir. Bu durumda ilgili kayıt ortak kullanılan bir kaynak olarak değerlendirilebilir.

Bilgisayar hafızası için de benzer durum geçerli olabilmektedir. Bilgisayar hafızasının tek olması ancak hafızaya erişen farklı iş parçacıklarının bulunabilmesi, bilgisayar hafızasını ortak kullanılan bir kaynak olarak değerlendirmek için yeterlidir.

Tekli İş Parçacık Uygulamaları ve Kaynak Paylaşımı

Tekli iş parçacık uygulamalarında, erişilen kaynağın ve erişen iş parçacığının tek olması, ortak kaynak kullanımı ve iş parçacıklarının senkronizasyonu konularını programlama yaparken göz ardı edebilmeyi sağlamaktadır.

Aşağıdaki örnekte statik bir alan olan Toplam değerine, KontrolEt isimli metot içerisinde değer atanmakta ve atanan değer kontrol edilmektedir.

C#

**class** Test {

**public** **static** **int** Toplam = 0;

}

**static** **void** KontrolEt() {

Test.Toplam = 12;

// Bu bölümde çeşitli işlemler yapıldığı varsayılmaktadır.

// Yapılan işlemlerin 10ms kadar sürdüğü değerlendirilerek  
 // iş parçacığı bekletilmektedir.

Thread.Sleep(10);

**int** sonuc = Test.Toplam;

**if** (sonuc == 12) {

Console.WriteLine("{0} iş parçacığı doğru değeri elde etti.",

Thread.CurrentThread.ManagedThreadId);

}

**else** {

Console.WriteLine("{0} numaralı iş parçacığı hatalı değer elde etti.",

Thread.CurrentThread.ManagedThreadId);

Console.WriteLine("{0} numaralı iş parçacığı değeri {1} olarak okudu",

Thread.CurrentThread.ManagedThreadId, sonuc);

}

}

**static** **void** Main(**string**[] args) {

KontrolEt();

}

VB.NET

**Class** Test

**Public** **Shared** Toplam **As** **Integer** = 0

**End** **Class**

**Shared** **Sub** KontrolEt()

Test.Toplam = 12

' Bu bölümde çeşitli işlemler yapıldığı varsayılmaktadır.

' Yapılan işlemlerin 10ms kadar sürdüğü değerlendirilerek  
 ' iş parçacığı bekletilmektedir.

Thread.Sleep(10)

**Dim** sonuc **As** **Integer** = Test.Toplam

**If** sonuc = 12 **Then**

Console.WriteLine("{0} iş parçacığı doğru değeri elde etti.", \_

Thread.CurrentThread.ManagedThreadId)

**Else**

Console.WriteLine("{0} numaralı iş parçacığı hatalı değer elde etti.", \_

Thread.CurrentThread.ManagedThreadId)

Console.WriteLine("{0} numaralı iş parçacığı değeri {1} olarak okudu", \_

Thread.CurrentThread.ManagedThreadId, sonuc)

**End** **If**

**End** **Sub**

**Shared** **Sub** Main(**ByVal** args() **As** **String**)

KontrolEt()

**End** **Sub**

Kod 245, Tekli İş Parçacık Uygulamaları ve Ortak Kaynak Kullanımı

Kod çıktısı beklenildiği gibi aşağıda gösterilmiştir.

10 iş parçacığı doğru değeri elde etti.

Alanın statik olarak tanımlanmış olması, aynı uygulama etki alanı içerisinde bu değerin paylaşılan bir kaynak olmasını beraberinde getirmektedir. Buna rağmen erişimin sadece tek noktadan, bir başka anlatımla erişimin sadece temel iş parçacığı tarafından yapılıyor olması, alan değerinin değiştirilmesi ve değerinin okunması sırasında herhangi bir ek işlem yapmayı gerektirmemektedir.

Örnekte alan değeri 12 olarak değiştirilmiş, bir sonraki satırda da değiştirildiği şekliyle başarıyla okunabilmiştir.

Çoklu İş Parçacık Uygulamaları ve Kaynak Paylaşımı

Çoklu iş parçacık uygulamalarında aynı kaynağa farklı iş parçacıklarının eş zamanlı erişimi mümkündür.

Kaynağın tek, kaynağa erişimin çok noktadan yapılıyor olması bu tür uygulamaların beklenildiği gibi çalışması için çeşitli önlemlerin alınmasını gerektirmektedir.

Aşağıdaki kod parçasında, bir önceki konuda gösterilen KontrolEt metodu tekrar çağrılmaktadır.

C#

**static** **void** Degistir() {

Console.WriteLine("Toplam değeri {0} tarafından değiştirildi.",   
 Thread.CurrentThread.ManagedThreadId);

Test.Toplam = 46;

}

**static** **void** Main(**string**[] args) {

KontrolEt();

ThreadStart ts = **new** ThreadStart(Degistir);

Thread t = **new** Thread(ts);

t.Start();

KontrolEt();

}

VB.NET

**Shared** **Sub** Degistir()

Console.WriteLine("Toplam değeri {0} tarafından değiştirildi.", \_  
 Thread.CurrentThread.ManagedThreadId)

Test.Toplam = 46

**End** **Sub**

**Shared** **Sub** Main(**ByVal** args() **As** **String**)

KontrolEt()

**Dim** ts **As** ThreadStart = **New** ThreadStart(**AddressOf** Degistir)

**Dim** t **As** Thread = **New** Thread(ts)

t.Start()

KontrolEt()

**End** **Sub**

Kod 246, Çoklu İş Parçacık Uygulamaları ve Ortak Kaynak Kullanımı

Kod çıktısı aşağıda gösterilmiştir.

10 iş parçacığı doğru değeri elde etti.

Toplam değeri 11 iş parçası tarafından değiştirildi.

10 numaralı iş parçacığı hatalı değer elde etti.

10 numaralı iş parçacığı değeri 46 olarak okudu.

KontrolEt isimli metot ilk defa çağrıldığında program beklenilen şekliyle çalışmaktadır. Daha sonra yeni bir iş parçacığı oluşturulmaktadır. Yeni oluşturulan iş parçacığı, Degistir isimli metodu çağırmakta ve Test.Toplam alan değeri değiştirilmektedir.

KontrolEt isimli metot içerisinde alan değeri değiştirilip hemen ardından tekrar okunmasına rağmen, çıktıda da gözüktüğü gibi, hatalı değer elde edilmektedir. Ortak kullanılan kaynak olan Test.Toplam alanının doğru paylaşılamaması, bir başka anlatımla temel iş parçacığı ile yeni oluşturulan iş parçacığının doğru senkronize edilememesi sonucu, atama / atanan değeri okuma işleminde dahi hatalı sonuç alınabilmektedir.

Yukarıdaki kod yapılan işlemler aşağıda sırayla gösterilmiştir.

1. Yeni oluşturulan iş parçacığı Thread.Start metodu ile çalışmaya başlamamış, sadece çalışma sinyali gönderilmiştir.
2. Hemen ardından temel iş parçacığı tarafından çağrılan KontrolEt isimli metot çağrılmış, bu metot içerisinde Toplam.Test alanının değeri 12 olarak değiştirilmiştir.
3. Thread.Sleep metodu ile 10ms kadar temel iş parçacığı bekletilmiştir.
4. Temel iş parçacığı beklerken, yeni oluşturulan iş parçacığı çalışmaya başlamış ve Test.Toplam alanının değerini 46 olarak değiştirmiştir.
5. 10ms sürenin tamamlanması ile temel iş parçacığı tekrar çalışmaya başlamıştır. Yerel sonuc isimli değişkene aktarılan değer, yeni oluşturulan iş parçacığı tarafından değiştirilen 46 değeri olmuştur.

Listelenen kaynak kodlar \Bölüm 15\SynProblem klasöründe bulunmaktadır.

Atomik Operasyonlar

Atomik operasyon kavramı, herhangi bir iş parçacığı tarafından tek adımda, bölünemez şekilde yapılan çalıştırma işlemini ifade etmektedir.

Atomik operasyonların kaynak kod içerisinde tek satır veya ifadeye karşı gelme zorunluluğu bulunmamaktadır. En basit şekliyle statik bir alanın değerinin değiştirilmesi atomik değildir.

C#

Test.Toplam = Test.Toplam + 1;

VB.NET

Test.Toplam = Test.Toplam + 1

Kod 247, Atomik Olmayan Operasyonlar

Örnekte statik bir alan olan Toplam değeri bir artırılmaktadır. Program açısından değerlendirildiğinde bölünemez bir parça gibi düşünülebilen bu işlemin atomik olmadığına dikkat edilmelidir. İşlemin tamamlanabilmesi için arka planda işlemci tarafından farklı alt işlemler yapılmaktadır.

Atomik operasyonların en önemli özelliği yapılan işlemin bölünemez olmasıdır. Atomik yazma işleminde, yazma işleminin tüm alt adımları başarıyla tamamlanana kadar diğer iş parçacıklarının değere erişimleri yasaklanmaktadır. Benzer şekilde atomik okuma işleminde, işlem tüm alt adımlarıyla tamamlanana kadar diğer iş parçacıklarının değeri değiştirmeleri mümkün değildir.

Yukarıdaki verilen basit değer yazma ve okuma örneğinde, aşağıdaki senaryonun çoklu iş parçacık uygulamalarında gerçekleşme olasılığı her zaman bulunmaktadır.

1. İlk iş parçacığı Test.Toplam + 1 anlatım değerini hesaplamak için Test.Toplam değerini okumaktadır. Örneğin ilk değer 10 olarak okunmaktadır.
2. İlk iş parçacığı henüz değer yazma adımına geçmeden, eş zamanlı olarak ikinci iş parçacığı Test.Toplam + 1 anlatım değerini hesaplamak için Test.Toplam değerini okumaktadır. Okunan değer yine 10 olmaktadır.
3. İlk iş parçacığı anlatım değerini 11 olarak hesaplamakta ve Test.Toplam değerini değiştirmektedir.
4. İkinci iş parçacığı benzer şekilde anlatım değerini 11 olarak hesaplamakta ve Test.Toplam değerini 11 olarak değiştirmektedir.
5. İki iş parçacığı Test.Toplam = Test.Toplam + 1 ifadesini çalıştırmış olmalarına rağmen Toplam değeri sadece bir artırılmıştır.

Yukarıda gösterilen senaryo çoklu iş parçacık uygulamalarında her zaman gerçekleşmeyebilir. Böylece, çoklu iş parçacık uygulaması bazen beklenilen sonucu üretirken bazen hatalı sonuç üretebilir. Bu durum, özellikle iş parçacık senkronizasyonunun doğru yapılmamasından kaynaklanan problemlerin teşhisini zorlaştırabilir.  
  
Çoklu iş parçacık uygulamaları geliştirilmeden önce, iş parçacık senkronizasyonu ve ortak kaynak paylaşımı konusu bütün kavramsal boyutlarıyla birlikte anlaşılmış olmalıdır. Aksi durumda program geliştirilirken yapılan testlerde doğru sonuçlar elde edilmiş olmasına rağmen, her an hatalı sonuç üretme riskine sahip yazılımlar ortaya çıkabilir.

İş Parçacık Senkronizasyon Yöntemleri

İş parçacıklarının senkronizasyonu ve ortak kaynakların doğru şekilde paylaşımının yapılabilmesi amacıyla .NET iş parçacık mimarisi tarafından çeşitli yöntemler sunulmaktadır.

Bu amaçla System.Threading ad alanı içerisinde kullanıma sunulan sınıflar, gerek CLR gerek işletim sistemi tarafından daha alt seviyede yapılabilecek senkronizasyon yöntemlerini içermektedir.

System.Threading.Interlocked Sınıfı

System.Threading.Interlocked sınıfı, birden çok iş parçacığı tarafından paylaşılan değişkenler için atomik operasyonlar sağlamaktadır.

Interlocked sınıfı statik metotları ve açıklamaları aşağıda gösterilmiştir.

| Üye Adı | Açıklama |
| --- | --- |
| Add | Parametre olarak iletilen iki değeri atomik operasyon olarak ekler ve sonucu döndürür. |
| CompareExchange | Parametre olarak iletilen değerlerin eşitliğini kontrol eder ve değerler eşit değil ise yer değiştirme yapar. |
| Decrement | Parametre olarak iletilen değeri atomik operasyon olarak bir eksiltir. |
| Exchange | Parametre olarak iletilen değerleri atomik operasyon olarak yer değiştirir. |
| Increment | Parametre olarak iletilen değeri atomik operasyon olarak bir artırır. |
| Read | 64Bit sayı değerini atomik operasyon olarak okur ve okuduğu değeri döndürür. Metot gereksinimi, 32Bit sistemlerde 64Bit değerlerin iki parça olarak ele alınmasından kaynaklanmaktadır. |

Tablo 95, System.Threading.Interlocked Sınıfı Üye Bilgileri

Aşağıdaki örnekte, bir önceki konuda aktarılan Test.Toplam alan değeri artırımının çoklu iş parçacık uygulamalarında Interlocked.Increment metodu ile doğru şekilde yapılması gösterilmiştir.

C#

// Test.Toplam = Test.Toplam + 1;

Interlocked.Increment(**ref** Test.Toplam);

VB.NET

' Test.Toplam = Test.Toplam + 1

Interlocked.Increment(Test.Toplam)

Kod 248, System.Threading.Interlocked Kullanım Örneği

Asal sayıların en hızlı şekilde Interlocked sınıfı ve iş parçacıkları kullanılarak bulunması mümkündür. Aşağıdaki örnekte 10000 sayısından küçük asal sayılar 10 adet iş parçacığı tarafından bulunmakta ve ekrana yazdırılmaktadır.

C#

**class** Test {

// Toplam asal sayı

**public** **static** **int** Toplam = 0;

// Maksimum asal sayı

**public** **const** **int** Max = 10000;

// Kontrol edilen sayı

**public** **static** **int** Sayi = Max;

}

**public** **static** **bool** AsalSayimi(**int** sayi) {

. . .

}

**static** **void** AsalBul() {

**int** sayi = Interlocked.Decrement(**ref** Test.Sayi);

**while** (sayi > 2) {

**if** (AsalSayimi(sayi)) {

Interlocked.Increment(**ref** Test.Toplam);

Console.WriteLine("{0}, iş parçacığı {1} tarafınfan bulundu.",   
 sayi,   
 Thread.CurrentThread.ManagedThreadId);

}

sayi = Interlocked.Decrement(**ref** Test.Sayi);

}

}

**static** **void** Main(**string**[] args) {

Thread[] list = **new** Thread[10];

. . .

**foreach** (Thread t **in** list)

t.Join();

Console.WriteLine("{0} değerinden küçük toplam {1} asal sayı vardır",  
 Test.Max,   
 Test.Toplam);

}

VB.NET

**Class** Test

' Toplam asal sayı

**Public** **Shared** Toplam **As** **Integer** = 0

' Maksimum asal sayı

**Public** **Shared** **ReadOnly** Max **As** **Integer** = 10000

' Kontrol edilen sayı

**Public** **Shared** Sayi **As** **Integer** = Max

**End** **Class**

**Public** **Shared** **Function** AsalSayimi(**ByVal** sayi **As** **Integer**) **As** **Boolean**

. . .

**End** **Function**

**Shared** **Sub** AsalBul()

**Dim** sayi **As** **Integer** = Interlocked.Decrement(Test.Sayi)

**While** sayi > 2

**If** AsalSayimi(sayi) **Then**

Interlocked.Increment(Test.Toplam)

Console.WriteLine("{0}, iş parçacığı {1} tarafınfan bulundu.", \_  
 sayi, \_  
 Thread.CurrentThread.ManagedThreadId)

**End** **If**

sayi = Interlocked.Decrement(Test.Sayi)

**End** **While**

**End** **Sub**

**Shared** **Sub** Main(**ByVal** args() **As** **String**)

**Dim** list(9) **As** Thread

. . .

**For** **Each** t **As** Thread **In** list

t.Join()

**Next**

Console.WriteLine("{0} değerinden küçük toplam {1} asal vardır.", \_  
 Test.Max, \_  
 Test.Toplam)

**End** **Sub**

Kod 249, System.Threading.Interlocked ve Asal Sayıların Bulunması

Örnekte, Interlocked.Decrement metodu ile Test.Sayi alan değeri bir azaltılmaktadır. Atomik olarak azaltılan değer yardımıyla her iş parçacığının tekil sayılar elde etmesi sağlanmaktadır. Daha sonra, eş zamanlı olarak on iş parçacığı tarafından asal sayı kontrolü yapılmaktadır.

Kontrol edilen sayının asal olması durumunda, Interlocked.Increment metodu ile atomik olarak Test.Toplam değeri artırılmaktadır. Böylece iş parçacıkları tarafından eş zamanlı bulunan asal sayı toplamı elde edilmektedir.

Listelenen kaynak kodlar \Bölüm 15\AsalSayilar klasöründe bulunmaktadır.

Senkronizasyon Kilit Blokları

System.Threading.Interlocked sınıfı, değişken değerleri için atomik yöntemler sağlamasına rağmen bazı durumlarda bu düzeyde senkronizasyon yeterli olmayabilir.

Senkronizasyon kilit blokları, blok içerisinde bulunan kodların eş zamanlı olarak sadece bir iş parçacığı tarafından çalıştırılmasının istenildiği durumlarda kullanılmalıdır. Bir başka anlatımla, senkronizasyon kilit blokları içerisinde bulunan kodlar atomik olarak değerlendirilmektedir.

lock ve SyncLock (sırasıyla C# ve VB.NET) rezerve kelimeleri senkronizasyon kilit bloğunu belirtmek için kullanılmaktadır.

| C# | VB.NET |
| --- | --- |
| **class** Islem {  **public** **void** IslemYap() {  **lock**(**this**) {  // Senkronize edilmesi istenen  // kod bloğu buraya   // yazılmalıdır.  }  }  } | **Class** Islem  **Public** **Sub** IslemYap()  **SyncLock** **Me**  ' Senkronize edilmesi istenen  ' kod bloğu buraya   ' yazılmalıdır.  **End** **SyncLock**  **End** **Sub**  **End** **Class** |

Kod 250, Senkronizasyon Kilit Blokları

Lock ve SyncLock rezerve kelimeleri, senkronizasyon kilidini tanımlamak amacıyla herhangi bir nesne gerektirmektedir. Örnekte nesnenin kendisi kilit tanımlayıcı olarak kullanılmıştır.

Kilit tanımlayıcılar, tek iş parçacığı tarafından bir bütün olarak çalıştırılması istenen farklı senkronizasyon blokları oluşturmak için kullanılmaktadır.

| C# | VB.NET |
| --- | --- |
| **class** Islem {  **object** kilit1 = **new** **object**();  **object** kilit2 = **new** **object**();    **public** **void** IslemYap() {  **lock**(kilit1) {  }  **lock** (kilit2) {  }  }  } | **Class** Islem  **Public** Kilit1 **As** **Object**= **New** **Object**()  **Public** Kilit2 **As** **Object**= **New** **Object**()  **Public** **Sub** IslemYap()  **SyncLock** Kilit1  **End** **SyncLock**  **SyncLock** Kilit1  **End** **SyncLock**  **End** **Sub**  **End** **Class** |

Kod 251, Senkronizasyon Kilit Tanımlayıcı Örnekleri

Örnekte, herhangi bir iş parçacığı tarafından ilk kilit tarafından ifade edilen senkronizasyon bloğunun çalıştırılması tamamlandıktan sonra, bir başka iş parçacığı Kilit1 ile ifade edilen bloğu çalıştırabilmektedir.

Birden çok iş parçacığı tarafından paylaşılan değişkenler için atomik operasyonlar sağlayan Interlocked sınıfı en hızlı senkronizasyonu sağlamaktadır. Interlocked sınıfı tarafından sunulan statik metotların yeterli olduğu durumlarda bu sınıfın kullanımı tercih edilmelidir.

Senkronizasyon kilit nesnesi olarak nesnenin kendisi (this, me) veya sınıf tip nesnesi (typeof, GetType) kullanmak yerine, yukarıdaki örnekte olduğu gibi gizli alanlar tanımlamak ve bu alanları kilit nesneleri olarak kullanmak önerilmektedir.

Monitör Sınıfı

Senkronizasyon kilit blokları ile yapılan iş parçacık senkronizasyonunu System.Threading.Monitor sınıfı ile yapmak mümkündür. Aşağıda gösterilen kod parçası birebir lock ve SyncLock blokları aynı anlamı taşımaktadır.

| C# | VB.NET |
| --- | --- |
| **class** Islem {  **public** **void** IslemYap() {  Monitor.Enter(**this**)*;*  **try** {  // Senkronize edilmesi istenen  // kod bloğu buraya   // yazılmalıdır.  }  **finally** {  Monitor.Exit(**this**)*;*  }  }  } | **Class** Islem  **Public** **Sub** IslemYap()  Monitor.Enter(**Me**)  **Try**  ' Senkronize edilmesi istenen  ' kod bloğu buraya   ' yazılmalıdır.  **Finally**  Monitor.Exit(**Me**)  **End** **Try**  **End** **Sub**  **End** **Class** |

Kod 252, System.Threading.Monitor Kullanım Örneği

Monitor.TryEnter metodu, parametre olarak iletilen nesne üzerindeki kilidi kontrol etmek amacıyla kullanılabilir. Metodun true değerini döndürmesi işlemin başarıyla tamamlandığını, false değerini döndürmesi işlemin başarısız olduğunu belirtmektedir.

Monitor sınıfının Wait ve Pulse metotları, iş parçacıkları sinyal mekanizması sağlamaktadır. Wait metodu, bir başka iş parçacığından sinyal alana kadar çalıştırmayı bloke etmekte, Pulse metodu beklenen sinyali iletmektedir. PulseAll metodu sinyal bekleyen bütün iş parçacıklarına sinyal iletmek için kullanılmaktadır.

Aşağıda Monitor.Wait ve Monitor.Pulse metotlarının kullanıldığı örnek sınıf gösterilmiştir.

C#

**class** Test {

**private** **object** sync = **new** **object**();

**private** **bool** sinyalGeldiMi;

**public** **void** IslemYap() {

**lock** (sync) {

Console.WriteLine("Sinyal bekleniyor ... {0}", sinyalGeldiMi);

Monitor.Wait(sync);

Console.WriteLine("Sinyal geldi ... {0}", sinyalGeldiMi);

}

}

**public** **void** SinyalIlet() {

**lock** (sync) {

sinyalGeldiMi = **true**;

Console.WriteLine("Sinyal iletiliyor ...");

Monitor.Pulse(sync);

}

}

}

VB.NET

**Class** Test

**Private** sync **As** **Object** = **New** **Object**()

**Private** sinyalGeldiMi **As** **Boolean**

**Public** **Sub** IslemYap()

**SyncLock** sync

Console.WriteLine("Sinyal bekleniyor ... {0}", sinyalGeldiMi)

Monitor.Wait(sync)

Console.WriteLine("Sinyal geldi ... {0}", sinyalGeldiMi)

**End** **SyncLock**

**End** **Sub**

**Public** **Sub** SinyalIlet()

**SyncLock** sync

sinyalGeldiMi = **True**

Console.WriteLine("Sinyal iletiliyor ...")

Monitor.Pulse(sync)

**End** **SyncLock**

**End** **Sub**

**End** **Class**

Kod 253, System.Monitor.Wait ve System.Monitor.Pulse Metotları Kullanımı

Örnek sınıfta, IslemYap metodu Monitor.Wait metodu ile sinyal gelmesini beklemektedir. Temel iş parçacığı tarafından çağrılan SinyalIlet metodu ile sinyal iletilmektedir.

C#

Test testObj = **new** Test();

**new** Thread(t.IslemYap).Start();

Thread.Sleep(1000);

t.SinyalIlet();

VB.NET

**Dim** testObj **As** Test = **New** Test()

**Dim** t **As** Thread = **New** Thread(**AddressOf** testObj.IslemYap)

t.Start()

Thread.Sleep(1000)

testObj.SinyalIlet()

Kod çıktısı aşağıda gösterilmiştir.

Sinyal bekleniyor ... False

Sinyal iletiliyor ...

Sinyal geldi ... True

Listelenen kaynak kodlar \Bölüm 15\System.Monitor klasöründe bulunmaktadır.

Uçuculuk Kavramı

Uçuculuk kavramı (ing. Volatility), temel tip değerlerinin okunması ve yazılması işleminde, değerlerin optimizasyon amacıyla CPU içerisinde kaşelenmesi (ing. Caching) ile ortaya çıkabilecek problemleri ifade etmektedir.

| C# | VB.NET |
| --- | --- |
| **static** **int** bolunen = 20, bolen = 0*;*  **static** void IslemYap() {  **while** (bolen == 0)  *;*  Console.WriteLine(bolunen / bolen)*;*  }  **static** **void** Main() {  **new** Thread(IslemYap).Start()*;*  bolunen = 10*;*  bolen = 5*;*  Console.ReadLine()*;*  } | **Shared** bolunen **As** **Integer** = 20  **Shared** bolen **As** **Integer** = 0  **Shared** **Sub** IslemYap()  **While** bolen = 0  **End** **While**  Console.WriteLine(bolunen \ bolen)  **End** **Sub**  **Shared** **Sub** Main()  **Dim** t **As** Thread  t = **New** Thread(**AddressOf** IslemYap)  t.Start()  bolunen = 10  bolen = 5  Console.ReadLine()  **End** **Sub** |

Kod 254, Uçuculuk Kavramı ve Örnek Kod Parçası

Örnekte, bolunen isimli alanın ilk değeri 20'dir ve daha sonra temel iş parçacığı tarafından alan değeri 10 olarak değiştirilmektedir. Örnek tek işlemcili sistemlerde her zaman 2 değerini ekrana yazarken, çok işlemcili sistemlerde 20 / 5 = 4 değerini yazma riski bulunmaktadır.

Risk sebebi, optimizasyon amacıyla işletim sistemi tarafından farklı işlemcilere farklı iş parçacıklarının yönlendirilmesi ile işlemci içerisinde değişken değerlerinin kaşelenmesidir.

bolunen isimli alan değeri temel iş parçacığı tarafından değiştirildikten sonra dahi tüm işlemciler tarafından alana atanan değer görülmeyebilir. Bu durumda, örneğin temel iş parçacığının üzerinde çalıştığı işlemci dışındaki işlemcilerde çalışan yeni iş parçacığı bu alanın eski değerini okuyabilir.

C# için volatile rezerve kelimesi bu problemi elimine etmek amacıyla kullanılmaktadır. Aynı işlev C# ve VB.NET için, Thread.VolatileRead ve Thread.VolatileWrite metotları ile çözümlenebilir. Her iki çözümde de alan değerinin işlemci tarafından kaşelenmesinin önüne geçilmektedir.

| C# | VB.NET |
| --- | --- |
| **volatile** **static** **int** bolunen = 20;  **volatile** **static** **int** bolen = 0;  **static** **void** Main() {  **new** Thread(IslemYap).Start();  bolunen = 10;  bolen = 5;  Console.ReadLine();  }  **static** **void** IslemYap() {  **while** (bolen == 0)  ;  Console.WriteLine(bolunen / bolen);  } | **Shared** bolunen **As** **Integer** = 20  **Shared** bolen **As** **Integer** = 0  **Shared** **Sub** Main()  **Dim** t **As** Thread  t = **New** Thread(**AddressOf** IslemYap)  t.Start()  Thread.VolatileWrite(bolunen, 10)  Thread.VolatileWrite(bolen, 5)  Console.ReadLine()  **End** **Sub**  **Shared** **Sub** IslemYap()  **While** Thread.VolatileRead(bolen) = 0  **End** **While**  Console.WriteLine(bolunen \ bolen)  **End** **Sub** |

Kod 255, Uçuculuk Örneği

volatile rezerve kelimesi ve Thread.VolatileRead / Thread.VolatileWrite metotları yerine Interlocked ve Monitor sınıfı veya senkronizasyon kilit blokları kullanımı önerilmektedir.

Okuyucu / Yazıcı Kilit Sınıfı

Programlama sırasında sık yapılan işlemlerden birisi, herhangi bir kaynağın değerinin okunması ve kaynağa değer yazılmasıdır. Okuma ve yazma işlemleri şu ana kadar aktarılan senkronizasyon yöntemleri ile düzenlenebilir. Örneğin her işlemde senkronizasyon kilit blokları kullanılabilir.

Okuma işleminin yazma işlemine göre çok daha sık yapıldığı durumlarda, senkronizasyon kilit blokları gereksiz yere kaynağa erişimi engelleyebilir. Bu durumda System.Threading.ReaderWriterLock sınıfı yardımıyla okuma ve yazma kilitleri birbirlerinden bağımsız olarak düzenlenebilir.

ReaderWriterLock sınıfının pratikte kullanım amacı, değeri birden çok iş parçacığı tarafından sıklıkla okunan, daha az sıklıkla sadece bir iş parçacığı tarafından değeri değiştirilen kaynakların paylaşımını yönetmektir.

Aşağıda ReaderWriterLock sınıfı temel üye bilgi ve açıklamaları gösterilmiştir.

| Üye Adı | Açıklama |
| --- | --- |
| IsReaderLockHeld | Aktif iş parçacığının okuma kilidini tutması durumunda true değerini döndürür. |
| IsWriterLockHeld | Aktif iş parçacığının yazma kilidini tutması durumunda true değerini döndürür. |
| AcquireReaderLock | Okuma kilidini elde eder. Parametre olarak iletilen zaman aşımı süresince kilidin elde edilememesi durumunda istisna (System.ApplicationException) fırlatılmaktadır. |
| AcquireWriterLock | Yazma kilidini elde eder. Parametre olarak iletilen zaman aşımı süresince kilidin elde edilememesi durumunda istisna (System.ApplicationException) fırlatılmaktadır. |
| ReleaseReaderLock | Elde edilen okuma kilidini serbest bırakır. |
| ReleaseWriterLock | Elde edilen yazma kilidini serbest bırakır. |

Tablo 96, System.Threading.ReaderWriterLock Sınıfı Üye Bilgileri

Aşağıdaki örnekte içerisinde tamsayılar bulunan liste üç farklı iş parçacığı tarafından kullanılmaktadır. Listeye rasgele sayılar eklenmekte, silinmekte ve bir başka iş parçacığı tarafından eş zamanlı listedeki elemanların toplamları ekrana yazdırılmaktadır.

C#

**static** ReaderWriterLock rw = **new** ReaderWriterLock();

**static** List<**int**> liste = **new** List<**int**>();

**static** Random rand = **new** Random();

**static** **void** ToplamYaz() {

**while** (**true**) {

rw.AcquireReaderLock(10000);

**int** toplam = 0;

**foreach** (**int** i **in** liste) toplam += i;

Console.WriteLine(toplam);

rw.ReleaseReaderLock();

}

}

**static** **void** ElemanEkle() {

**while** (**true**) {

rw.AcquireWriterLock(10000);

liste.Add(rand.Next(100));

Thread.Sleep(1);

rw.ReleaseWriterLock();

}

}

**static** **void** ElemanSil() {

**while** (**true**) {

rw.AcquireWriterLock(10000);

**if** (liste.Count > 0)

liste.Remove(rand.Next(100));

rw.ReleaseWriterLock();

}

}

**static** **void** Main(**string**[] args) {

**new** Thread(ElemanEkle).Start();

**new** Thread(ElemanSil).Start();

**new** Thread(ToplamYaz).Start();

}

VB.NET

**Private** **Shared** rw **As** ReaderWriterLock = **New** ReaderWriterLock()

**Private** **Shared** liste **As** List(**Of** **Integer**) = **New** List(**Of** **Integer**)()

**Private** **Shared** rand **As** Random= **New** Random()

**Private** **Shared** **Sub** ToplamYaz()

**While** **True**

rw.AcquireReaderLock(10000)

**Dim** toplam **As** **Integer** = 0

**For** **Each** i **As** **Integer** **In** liste

toplam = toplam + i

**Next**

Console.WriteLine(toplam)

rw.ReleaseReaderLock()

**End** **While**

**End** **Sub**

**Private** **Shared** **Sub** ElemanEkle()

**While** **True**

rw.AcquireWriterLock(10000)

liste.Add(rand.**Next**(100))

Thread.Sleep(1)

rw.ReleaseWriterLock()

**End** **While**

**End** **Sub**

**Private** **Shared** **Sub** ElemanSil()

**While** **True**

rw.AcquireWriterLock(10000)

**If** (liste.Count > 0) **Then**

liste.Remove(rand.**Next**(100))

**End** **If**

rw.ReleaseWriterLock()

**End** **While**

**End** **Sub**

**Shared** **Sub** Main(**ByVal** args() **As** **String**)

**Dim** t1 **As** Thread = **New** Thread(**AddressOf** ElemanEkle)

**Dim** t2 **As** Thread = **New** Thread(**AddressOf** ElemanSil)

**Dim** t3 **As** Thread = **New** Thread(**AddressOf** ToplamYaz)

t1.Start()

t2.Start()

t3.Start()

**End** **Sub**

Kod 256, System.Threading.ReaderWriterLock Kullanım Örneği

Zaman aşımı süresi olarak sonsuz değerini ifade eden -1 veya Timeout.Infinite özelliği kullanılabilir.

İş Parçacık Veri Slotları

Her iş parçacık nesnesi kendisine özel veri slotlarına sahip olabilmektedir. İş parçacığının veri slotuna yazılan veri sadece ilgili iş parçacığına özel olmaktadır.

Thread.GetData ve Thread.SetData metotları iş parçacık slotlarından veri okumak ve istenilen slota veri yazmak için kullanılmaktadır.

C#

**class** Test {

**static** **int** IsparcacikVerisi {

**get** {

LocalDataStoreSlot slot = Thread.GetNamedDataSlot("Veri");

**object** o = Thread.GetData(slot);

**if** (o == **null**) **return** 0;

**else return** (**int**)o;

}

**set** {

LocalDataStoreSlot slot = Thread.GetNamedDataSlot("Veri");

Thread.SetData(slot, value);

}

}

}

VB.NET

**Class** Test

**Shared** **Property** IsparcacikVerisi() **As** **Integer**

**Get**

**Dim** slot **As** LocalDataStoreSlot = Thread.GetNamedDataSlot("Veri")

**Dim** o **As** **Object** = Thread.GetData(slot)

**If** o **Is** **Nothing** **Then**

**Return** 0

**Else**

**Return** **CType**(o, **Integer**)

**End** **If**

**End** **Get**

**Set**(**ByVal** value **As** **Integer**)

**Dim** slot **As** LocalDataStoreSlot = Thread.GetNamedDataSlot("Veri")

Thread.SetData(slot, value)

**End** **Set**

**End** **Property**

**End** **Class**

Kod 257, Thread.GetData ve Thread.SetData Metotları Kullanım Örneği

Thread.FreeNamedDataSlot metodu veri slotunu serbest bırakmak için kullanılmaktadır. Farklı LocalDataStoreSlot nesneleri ile sadece ilgili iş parçacığına özgü verilerin saklanması mümkündür.

System.ThreadStaticAttribute öz niteliği, statik alan değerlerinin her iş parçacığı için farklı değerler alması gerektiğini belirtmek için kullanılabilir.   
  
Statik alanın bu öznitelik ile donatılması yukarıda gösterilen işlevi sağlamaktadır.

Diğer Senkronizasyon Yöntemleri

Farklı uygulama etki alanları içerisinde çalışan iş parçacıklarının da senkronize edilebilmesi amacıyla Windows işletim sistemi tarafından çeşitli senkronizasyon yöntemleri sağlanmaktadır.

Aşağıda Windows işletim sistemi tarafından sağlanan senkronizasyon yöntemleri, .NET iş parçacık mimarisi sınıf ve açıklamaları gösterilmiştir.

| Sınıf | Açıklama |
| --- | --- |
| Mutex | Farklı uygulama etki alanı ve prosesler içerisinde bulunan iş parçacıkları senkronizasyonunu sağlamak amacıyla kullanılmaktadır. |
| Semaphore | Farklı uygulama etki alanı ve prosesler içerisinde bulunan iş parçacıklarının istenilen kaynağa erişimlerini denetlemek için kullanılmalıdır. |
| EventWaitHandle | Farklı uygulama etki alanı ve prosesler içerisinde bulunan iş parçacıklarının birbirleriyle haberleşmelerini sağlamak amacıyla kullanılmalıdır. |

Tablo 97, Windows İşletim Sistemi İş Parçacık Senkronizasyon Bilgileri

Mutex, Semaphore ve EventWaitHandle sınıfları soyut bir sınıf olan System.Threading.WaitHandle sınıfını miras almaktadır. AutoResetEvent ve ManualResetEvent sınıfları EventWaitHandle sınıfını miras almaktadır.

Mutex sınıfı senkronizasyon kilit blokları ile aynı işlevlere sahiptir. Senkronizasyon kilit bloklarından en önemli farkı, farklı prosesler içerisinde bulunan iş parçacıklarına verilen destektir.

Örneğin Mutex sınıfı, uygulamanın birden fazla çalışmasını engellemek amacıyla kullanılabilir.

C#

**static** **void** Main(**string**[] args) {

// Monitor.Enter metoduna benzer şekilde yeni bir kilit ekle

Mutex m = **new** Mutex(**false**, "Profesyonel Programlama Teknikleri");

// Kilidi 1sn süreyle bekle.

**if** (!m.WaitOne(1000, **false**)) {

Console.WriteLine("Uygulama zaten çalışmakta");

**return**;

}

Console.WriteLine("Uygulama çalıştı ...");

Console.ReadLine();

m.ReleaseMutex();

}

VB.NET

**Shared** **Sub** Main(**ByVal** args() **As** **String**)

' Monitor.Enter metoduna benzer şekilde yeni bir kilit ekle

**Dim** m **As** Mutex = **New** Mutex(**False**, "Profesyonel Programlama Teknikleri")

' Kilidi 1sn süreyle bekle.

**If** **Not** m.WaitOne(1000, **False**) **Then**

Console.WriteLine("Uygulama zaten çalışmakta.")

**Return**

**End** **If**

Console.WriteLine("Uygulama çalıştı ...")

Console.ReadLine()

m.ReleaseMutex()

**End** **Sub**

Kod 258, System.Threading.Mutex Kullanım Örneği

Örnekte "Profesyonel Programlama Teknikleri" değeri ile tanımlanan mutex nesnesi oluşturulmaktadır. WaitHandle.WaitOne metodu çağrılarak mutex nesnesi kilidi 1sn süreyle beklenmektedir. WaitOne metodunun false değerini döndürmesi, daha önce bu metot ile bir kilit oluşturulduğu anlamını taşımaktadır.

Semaphore sınıfı, istenilen sayıda oluşturulabilen kapasite değeri ile, bir kaynağa eş zamanlı kaç iş parçacığının erişebileceğini denetlemek için kullanılmaktadır.

C#

// Maksimum 10 kapasite değeri ile semaphore nesnesi oluşturulmaktadır.  
// Başlangıçta 3 yer serbest bırakılmaktadır.

**static** Semaphore s = **new** Semaphore(3, 10);

**static** **int** aktif = 0;

**static** **object** sync = **new** **object**();

**static** **void** Main() {

// 100 adet iş parçacık nesnesi oluşturulmaktadır.

**for** (**int** i = 0; i < 100; i++) **new** Thread(IslemYap).Start();

Thread.Sleep(5000);

// 5 adet yer serbest bırakılmaktadır. Böylece semaphore nesnesi en fazla  
 // 8 adet erişime izin vermektedir.

s.Release(5);

}

**static** **void** IslemYap() {

// Giriş yapıldığı semaphore nesnesine bildirilmektedir.

// Bir başka anlatımla, semaphore nesnesi içerisinde serbest olan   
 // bir bölüm doldurulmaktadır.

s.WaitOne();

// Bu bölüme başta en fazla 3, program çalıştıktan 5sn sonra en fazla  
 // 8 adet iş parçacığı erişebilir.

**lock** (sync) {

Console.WriteLine(++aktif);

}

Thread.Sleep(1000);

Interlocked.Decrement(**ref** aktif);

s.Release();

}

VB.NET

' Maksimum 10 kapasite değeri ile semaphore nesnesi oluşturulmaktadır.  
' Başlangıçta 3 yer serbest bırakılmaktadır.

**Shared** s **As** Semaphore = **New** Semaphore(3, 10)

**Shared** aktif **As** **Integer** = 0

**Shared** sync **As** **Object** = **New** **Object**()

**Shared** **Sub** Main()

' 100 adet iş parçacık nesnesi oluşturulmaktadır.

**For** i **As** **Integer** = 0 **To** 99

**Dim** t **As** Thread = **New** Thread(**AddressOf** IslemYap)

t.Start()

**Next**

Thread.Sleep(5000)

' 5 adet yer serbest bırakılmaktadır. Böylece semaphore nesnesi en fazla  
' 8 adet erişime izin vermektedir.

s.Release(5)

**End** **Sub**

**Shared** **Sub** IslemYap()

' Giriş yapıldığı semaphore nesnesine bildirilmektedir.

' Bir başka anlatımla, semaphore nesnesi içerisinde serbest olan   
 ' bir bölüm doldurulmaktadır.

s.WaitOne()

' Bu bölüme başta en fazla 3, program çalıştıktan 5sn sonra en fazla  
 ' 8 adet iş parçacığı erişebilir.

**SyncLock** sync

aktif = aktif + 1

Console.WriteLine(aktif)

**End** **SyncLock**

Thread.Sleep(1000)

Interlocked.Decrement(aktif)

s.Release()

**End** **Sub**

Kod 259, System.Threading.Semaphore Kullanım Örneği

EventWaitHandle sınıfı, iş parçacıkları arasında sinyal mekanizması için kullanılmaktadır. Aşağıdaki örnekte temel iş parçacık nesnesi sinyali iletene kadar yeni oluşturulan iş parçası bloke olmaktadır.

C#

**static** EventWaitHandle h = **new** AutoResetEvent(**false**);

**static** **void** Main() {

**new** Thread(IslemYap).Start();

Thread.Sleep(1000);

Console.WriteLine("Sinyal iletiliyor ...");

h.Set();

}

**static** **void** IslemYap() {

Console.WriteLine("Sinyal bekleniyor ...");

h.WaitOne();

Console.WriteLine("Sinyal geldi.");

}

VB.NET

**Shared** h **As** EventWaitHandle = **New** AutoResetEvent(**False**)

**Shared** **Sub** Main()

**Dim** t **As** Thread = **New** Thread(**AddressOf** IslemYap)

t.Start()

Thread.Sleep(1000)

Console.WriteLine("Sinyal iletiliyor ...")

h.Set()

**End** **Sub**

**Shared** **Sub** IslemYap()

Console.WriteLine("Sinyal bekleniyor ...")

h.WaitOne()

Console.WriteLine("Sinyal geldi.")

**End** **Sub**

Kod 260, System.Threading.EventWaitHandle Kullanım Örneği

İş parçacık senkronizasyonu ve ortak kaynak paylaşımının tüm detaylarıyla anlaşılması ve pratikte doğru şekliyle kullanımı, bu konuda belirli bir deneyim seviyesine sahip olmayı gerektirmektedir. Sonraki konuya geçmeden önce, kitapta şu ana kadar sunulan bilgi ve örneklerin kısa bir tekrarı ve pratik yapılması önerilmektedir.

İş Parçacık Havuzu ve Zamanlayıcı Nesneleri

İş parçacıkları konusunda son olarak iş parçacık havuzu (ing. Thread Pool) ve zamanlayıcı (ing. Timer) nesneleri hakkında bilgiler verilecektir.

İş Parçacık Havuzu Kullanımı

System.Threading.ThreadPool sınıfı, .NET iş parçacık mimarisi tarafından sağlanan iş parçacık havuz sınıfıdır. İş parçacık havuzu içerisinde kullanıma hazır iş parçacıklarının bulunması, her iş parçacık nesnesinin oluşturulması adımında gereken kaynak gereksinimini elimine etmektedir.

Sağlanan bir diğer avantaj, uygulamanın sayısı belirli olan iş parçacık nesnelerini anında kullanabilmesi, nesne sayısının yetersiz olduğu durumda havuz kuyruğuna kolaylıkla yeni işleri gönderilebilmesidir.

Aşağıda ThreadPool sınıfının önemli görülen üyeleri ve açıklamaları gösterilmiştir.

| Üye Adı | Açıklama |
| --- | --- |
| GetAvailableThreads | Havuz içerisinde bulunan kullanıma hazır iş parçacık sayısını döndürür. |
| GetMaxThreads | Havuz tarafından desteklenebilecek en yüksek iş parçacık sayısını döndürür. |
| GetMinThreads | Havuz tarafından hazırda tutulan en az iş parçacık sayısını döndürür. |
| QueueUserWorkItem | Havuza iş göndermek için kullanılan metottur. Metodun true döndürmesi işin başarıyla kuyruğa atıldığını, false döndürmesi kuyruğa atmanın başarısız olduğu anlamına gelmektedir. |
| RegisterWaitForSingleObject | Sinyal iletildiğinde çalıştırılması istenilen işi havuza göndermek için kullanılan metottur. |
| SetMaxThreads | Maksimum iş parçacık sayısını değiştirmek için kullanılan metottur. |
| SetMinThreads | Minimum iş parçacık sayısını değiştirmek için kullanılan metottur. |

Tablo 98, System.Threading.ThreadPool Sınıfı Üye Bilgileri

Kullanım Örnekleri

ThreadPool.QueueUserWorkItem ve ThreadPool.RegisterWaitForSingleObject metotları havuza iş göndermek için kullanılan metotlardır.

Aşağıdaki örnek kod parçasında QueueUserWorkItem metodu kullanılarak havuza iş gönderilmesi gösterilmiştir. Gönderilen iş, havuz içerisindeki herhangi bir iş parçacığı tarafından yapılmaktadır.

C#

**static** **void** IslemYap(**object** veri) {

Console.WriteLine("Veri: {0}, İş Parçacığı: {1}",   
 veri,   
 Thread.CurrentThread.ManagedThreadId);

Thread.Sleep(1000);

}

**public** **static** **void** Main() {

Console.WriteLine("Temel iş parçacık no: {0}",   
 Thread.CurrentThread.ManagedThreadId);

WaitCallback callBack = **new** WaitCallback(IslemYap);

**if** (!ThreadPool.QueueUserWorkItem(callBack, "Test"))

Console.WriteLine("Kuyruğa atma başarısız ...");

Console.ReadLine();

}

VB.NET

**Shared** **Sub** IslemYap(**ByVal** veri **As** **Object**)

Console.WriteLine("Veri: {0}, İş Parçacığı: {1}", \_  
 veri, \_  
 Thread.CurrentThread.ManagedThreadId)

Thread.Sleep(1000)

**End** **Sub**

**Shared** **Sub** Main()

Console.WriteLine("Temel iş parçacık no: {0}", \_  
 Thread.CurrentThread.ManagedThreadId)

**Dim** callBack **As** WaitCallback = **New** WaitCallback(**AddressOf** IslemYap)

**If** **Not** ThreadPool.QueueUserWorkItem(callBack, "Test") **Then**

Console.WriteLine("Kuyruğa atma başarısız ...")

**End** **If**

Console.ReadLine()

**End** **Sub**

Kod 261, System.Threading.ThreadPool.QueueUserWorkItem Kullanımı

Kod çıktısı aşağıda gösterilmiştir.

Temel iş parçacık no: 11

Veri: Test, İş Parçacığı: 4

IslemYap metodunun iş parçacık havuzuna gönderilmesi ile metot, havuz tarafından sağlanan herhangi bir iş parçacık nesnesi ile çalıştırmaktadır.

Aşağıdaki örnekte ThreadPool.QueueUserWorkItem metodu ile Monitor.Wait ve Monitor.Pulse metotları kullanılarak iş parçacık havuzuna iş gönderilmesi ve gönderilen işler tamamlanana kadar temel iş parçacığının beklemesi gösterilmiştir.

C#

**private static object** sync = new **object**();

**private static int** calisanIsParcacikSayisi = 100;

**static** **void** IslemYap(**object** veri) {

Console.WriteLine("Veri: {0}, İş Parçacığı: {1}",   
 veri,   
 Thread.CurrentThread.ManagedThreadId);

Thread.Sleep(1000);

// Çalışan iş parçacık sayısı değerini azalt ve sinyal bekleyen  
 // iş parçacığına sinyali ilet.

**lock** (sync) {

calisanIsParcacikSayisi--;

Monitor.Pulse(sync);

}

}

**public** **static** **void** Main() {

WaitCallback callBack = **new** WaitCallback(IslemYap);

**for** (**int** i = 0; i < calisanIsParcacikSayisi; i++)

ThreadPool.QueueUserWorkItem(callBack, i);

Console.WriteLine("İşlerin tamamlanması bekleniyor ...");

// İşlemler tamamlanana kadar bekle.

**lock** (sync) {

**while** (calisanIsParcacikSayisi > 0)

Monitor.Wait(sync);

}

Console.WriteLine("İşler tamamlandı ...");

}

VB.NET

**Shared** sync **As** **Object** = **New** **Object**

**Shared** calisanIsParcacikSayisi **As** **Integer** = 100

**Private** **Shared** **Sub** IslemYap(**ByVal** veri **As** **Object**)

Console.WriteLine("Veri: {0}, İş Parçacığı: {1}", \_  
 veri, \_  
 Thread.CurrentThread.ManagedThreadId)

Thread.Sleep(1000)

' Çalışan iş parçacık sayısı değerini azalt ve sinyal bekleyen  
 ' iş parçacığına sinyali ilet.

**SyncLock** sync

calisanIsParcacikSayisi = calisanIsParcacikSayisi - 1

Monitor.Pulse(sync)

**End** **SyncLock**

**End** **Sub**

**Public** **Shared** **Sub** Main()

**Dim** callBack **As** WaitCallback = **New** WaitCallback(**AddressOf** IslemYap)

**For** I **As Integer** = 0 to calisanIsParcacikSayisi – 1

ThreadPool.QueueUserWorkItem(callBack, i)

**Next**

Console.WriteLine("İşlerin tamamlanması bekleniyor ...")

' İşlemler tamamlanana kadar bekle.

**SyncLock** sync

**While** (calisanIsParcacikSayisi > 0)

Monitor.Wait(sync)

**End** **While**

**End** **SyncLock**

Console.WriteLine("İşler tamamlandı ...")

**End** **Sub**

Kod 262, İleri Seviye İş Parçacık Havuzu ve Senkronizasyon Kullanımı

Listelenen kaynak kodlar \Bölüm 15\ThreadPool klasöründe bulunmaktadır.

ThreadPool.RegisterWaitForSingleObject metodu, havuza gönderilen işin bir sinyal iletildiğinde çalışmasını sağlamak için kullanılmaktadır.

C#

**static** ManualResetEvent sinyal = **new** ManualResetEvent(**false**);

**static** **void** Main() {

ThreadPool.RegisterWaitForSingleObject(sinyal, IslemYap, "test", -1, **true**);

Console.WriteLine("Sinyal iletiliyor ...");

sinyal.Set();

Console.ReadLine();

}

**static** **void** IslemYap(**object** data, **bool** timedOut) {

Console.WriteLine(data);

}

VB.NET

**Shared** sinyal **As** ManualResetEvent = **New** ManualResetEvent(**False**)

**Shared** **Sub** Main()

ThreadPool.RegisterWaitForSingleObject(sinyal, **AddressOf** IslemYap, \_  
 "test", -1, **True**)

Thread.Sleep(3000)

Console.WriteLine("Sinyal iletiliyor ...")

sinyal.Set()

Console.ReadLine()

**End** **Sub**

**Shared** **Sub** IslemYap(**ByVal** data **As** **Object**, **ByVal** timedOut **As** **Boolean**)

Console.WriteLine(data)

**End** **Sub**

Kod 263, ThreadPool.RegisterWaitForSingleOject Kullanım Örneği

Çoklu iş parçacık uygulamalarında, sağlamış olduğu kullanım kolaylığı, proses bazında sabit iş parçacık sayısı ve kuyruk desteği ile iş parçacık havuzu kullanımının tercih edilmesi önerilmektedir.  
  
Örneğin ASP.NET, sunucuya gelen isteklerin karşılanabilmesi amacıyla iş parçacık havuzu modelini kullanmaktadır. Gelen her bir istek için ayrı bir iş parçacık nesnesi oluşturmak yerine, ASP.NET tarafından yönetilen iş parçacık havuzu içerisinden hazır bir iş parçacığı isteğin karşılanması için tahsis edilmektedir. İsteğe yanıt verilmesi ve sonucun iletmesi ardından iş parçacığı tekrar kullanılmak üzere havuza geri iletilmektedir.

Zamanlayıcı Kullanımı

System.Threading.Timer sınıfı belirli zaman aralıklarında istenilen metodun çağrılması için kullanılmaktadır.

Aşağıda Timer kullanım örneği gösterilmiştir.

C#

**public** **static** **void** Main() {

TimerCallback callBack = **new** TimerCallback(IslemYap);

// 3 sn sonra çağırmaya başla ve 1 sn aralıklarla IslemYap isimli  
 // metodu çağır.

Timer t = **new** Timer(callBack, "Test", 3000, 1000);

Console.ReadLine();

t.Dispose();

}

**public** **static** **void** IslemYap(**object** data) {

Console.WriteLine(data);

}

VB.NET

**Shared** **Sub** Main()

**Dim** callBack **As** TimerCallback = **New** TimerCallback(**AddressOf** IslemYap)

' 3 sn sonra çağırmaya başla ve 1 sn aralıklarla IslemYap isimli  
 ' metodu çağır.

**Dim** t **As** Timer = **New** Timer(callBack, "Test", 3000, 1000)

Console.ReadLine()

t.Dispose()

**End** **Sub**

**Shared** **Sub** IslemYap(**ByVal** data **As** **Object**)

Console.WriteLine(data)

**End** **Sub**

Kod 264, System.Threading.Timer Kullanım Örneği

.NET sınıf kütüphanesi içerisinde farklı zamanlayıcı sınıfları bulunmaktadır.  
  
System.Threading.Timer bu konuda anlatılan sınıftır.  
  
System.Timers.Timer sınıfı, arka planda System.Threading.Timer sınıfını kullanan ve VS.NET tasarım desteği olan sınıftır.  
  
System.Windows.Forms.Timer sınıfı, WM\_TIMER Windows mesajı ile çalışan sınıftır.

Cehaletle deha arasındaki gerçek fark nedir biliyor musunuz ? Dehanın sınırları var cehaletinse hiçbir sınırı yoktur.

Whoopi Goldberg

Bölüm

16

Asenkron Programlama Modeli

A

senkron programlama modeli (ing. Asynchronous Programming Model - APM) .NET sınıf kütüphanesi içerisinde bulunan bir çok sınıf tarafından desteklenmektedir. APM yapısı, farklı işlemlerin farklı iş parçacıkları tarafından yapılmasını sağlaması ile, daha performanslı çalışan uygulama yazılımları elde etmek için kullanılabilmektedir. APM, programcı tarafından yeni iş parçacıkları oluşturularak sağlanan çoklu işletime önemli bir alternatif sunmaktadır.

.NET APM alt yapısı tarafından sunulan APM metot ve delegeleri ile, programcının yeni iş parçacık nesneleri oluşturması gerekmeden, istenilen metotlar farklı iş parçacıkları tarafından çalıştırılabilmektedir.

APM Temelleri

APM, desteklenen sınıflarda, istenilen metotların farklı iş parçacıkları tarafından çalıştırılabilmesini sağlamaktır. Genel standart olarak APM metot isimleri, Begin ve End ön ekleri ile donatılmaktadır.

Örneğin FileStream akım sınıfı, akımdan veri okumak ve yazmak için sırasıyla FileStream.Read ve FileStream.Write metotlarını sunmaktadır. Sunulan bu metotları senkron olarak nitelemek mümkündür. Senkron kavramı ile ifade edilmek istenen, metodu çağıran iş parçacığı tarafından çalıştırılması ve metot tarafından yapılan işlemlerin bitmesi sonucunda metodun çalışmasının tamamlanmasıdır. Aynı işlevi sağlayan FileStream.BeginRead ve FileStream.EndRead metotları, senkron olan metotların asenkron sürümleridir. Asenkron kavramı, metodun içerisinde yeni bir iş parçacığı oluşturulduğunu ve işlemlerin bu iş parçacığına atandığını belirtmektedir.

Asenkron metotların Begin ön eki ile donatılmış sürümleri, metot tarafından yapılması istenen işlemi yeni bir iş parçacığına atamakta ve hemen ardından metodun çalışması tamamlanmaktadır. End ön eki ile donatılmış sürümler, işlem henüz tamamlanmadı ise, yeni iş parçacığına atanmış işin tamamlanmasını beklemektedirler.

C#

FileStream f = **new** FileStream(@"c:\xxx.dat", FileMode.Open);

**byte** [] buffer = **new** **byte**[100];

f.Read(buffer, 0, buffer.Length);

// buffer isimli dizi içerisine okuma işlemi tamamlandığında   
// FileStream.Read metodunun çalışması tamamlanmış olmaktadır.

VB.NET

**Dim** f **As** FileStream = **New** FileStream("c:\xxx.dat", FileMode.Open)

**Dim** buffer() **As** **Byte** = **New** **Byte**(100) {}

f.Read(buffer, 0, buffer.Length)

' buffer isimli dizi içerisine okuma işlemi tamamlandığında   
' FileStream.Read metodunun çalışması tamamlanmış olmaktadır.

Kod 265, Senkron Programlama Örneği

Yukarıdaki örnekte FileStream.Read metodu içerisindeki kod temel iş parçacığı (veya bu metodu çağıran iş parçacığı) tarafından çalıştırılmaktadır. Örneğin dosyadan okuma işleminin 100ms kadar sürmesi durumunda, bu süre kadar çağıran iş parçacığı bloke olmaktadır.

C#

FileStream f = **new** FileStream(@"c:\xxx.dat", FileMode.Open);

**byte** [] buffer = **new** **byte**[100];

IAsyncResult sonuc = f.BeginRead(buffer, 0, buffer.Length, **null**, **null**);

// Dosyadan okuma işlemi FileStream.Read metodu tarafından yeni bir iş   
// parçacığına atanmış ve metodun çalışması tamamlanmıştır.

// Bu bölümde okuma işlemi devam ederken başka işlemler yapılabilir.

// Okuma işlemi EndRead metodu çağrılarak tamamlanmaktadır. EndRead metodu,  
// henüz okuma işlemi tamamlanmadı ise, işlem tamamlanana kadar çağıran iş   
// parçacığını bloke etmektedir.

f.EndRead(sonuc);

VB.NET

**Dim** f **As** FileStream = **New** FileStream("c:\xxx.dat", FileMode.Open)

**Dim** buffer(100) **As** **Byte**

**Dim** sonuc **As** IAsyncResult = f.BeginRead(buffer, 0, buffer.Length, **Nothing**, \_  
 **Nothing**)

' Dosyadan okuma işlemi FileStream.Read metodu tarafından yeni bir iş   
' parçacığına atanmış ve metodun çalışması tamamlanmıştır.

' Bu bölümde okuma işlemi devam ederken başka işlemler yapılabilir.

' Okuma işlemi EndRead metodu çağrılarak tamamlanmaktadır. EndRead metodu,  
' henüz okuma işlemi tamamlanmadı ise, işlem tamamlanana kadar çağıran iş   
' parçacığını bloke etmektedir.

f.EndRead(sonuc)

Kod 266, Asenkron Programlama Örneği

Okuma işleminin asenkron olarak yapıldığı yukarıdaki örnekte, FileStream.BeginRead metodu tarafından dosyanın okunması için yeni bir iş parçacığı oluşturulmaktadır. Böylece, bu metodu çağıran iş parçacığı bloke olmamakta, dosya okuma işlemi farklı bir iş parçacığı tarafından yapılırken diğer işlemler çağıran iş parçacığı tarafından paralel olarak sürdürülebilmektedir.

APM Metotları Kullanımı

System.IAsyncResult arabirimi, APM yapısında kullanılan en önemli arabirimdir.

C#

**public** **interface** IAsyncResult {

**object** AsyncState { **get**; }

WaitHandle AsyncWaitHandle { **get**; }

**bool** CompletedSynchronously { **get**; }

**bool** IsCompleted { **get**; }

}

VB.NET

**Public** **Interface** IAsyncResult

**ReadOnly** **Property** AsyncState **As** **Object**

**ReadOnly** **Property** AsyncWaitHandle **As** WaitHandle

**ReadOnly** **Property** CompletedSynchronously **As** **Boolean**

**ReadOnly** **Property** IsCompleted **As** **Boolean**

**End** **Interface**

Kod 267, System.IAsyncResult Arabirim Tanımı

IAsyncResult arabirim gerçekleştirimini yapan nesne BeginXXX asenkron metotları tarafından döndürülmektedir. Döndürülen nesne EndXXX metoduna parametre olarak iletilmekte, böylece asenkron işlem tamamlanmaktadır.

Asenkron programlama modeli gerçekleştirimi üç şekilde yapılabilir.

Bunlar

1. İşlem tamamlanana kadar bekleme
2. İşlem durumunu kontrol etme
3. İşlem tamamlandığında haberdar edilme

Şeklindedir.

İşlem Tamamlanana Kadar Bekleme Modeli

İşlem tamamlanana kadar bekleme modeli, asenkron işlemin BeginXXX metodu ile başlatılması, EndXXX metodu çağrısı ile işlemin tamamlanmasını beklemeyi içermektedir.

Bir önceki konuda gösterilen kod örneğinde bu model kullanılmıştır.

İşlem Durumunu Kontrol Etme

IAsyncResult.IsCompleted özelliğinin true değeri döndürmesi asenkron işlemin tamamlandığı anlamını taşımaktadır. İşlem durumunu kontrol etme modeli, bu özelliğin döndürdüğü değeri kontrol etmeyi temel almaktadır.

C#

IAsyncResult sonuc = f.BeginRead(buffer, 0, buffer.Length, **null**, **null**);

**while** (!sonuc.IsCompleted) {

// Bu bölümde işlem tamamlanana kadar diğer işler yapılabilir.

Thread.Sleep(10);

}

VB.NET

IAsyncResult sonuc = f.BeginRead(buffer, 0, buffer.Length, null, null)*;*

**while** (!sonuc.IsCompleted) {

' Bu bölümde işlem tamamlanana kadar diğer işler yapılabilir.

Thread.Sleep(10)*;*

}

Kod 268, Asenkron İşlem Durumunu Kontrol Etme Modeli

İşlem Tamamlandığında Haberdar Edilme

İşlem tamamlandığında haberdar edilme modeli, asenkron işlemin tamamlandığı durumda istenilen metodun çağrılmasını ifade etmektedir.

C#

**static byte**[] buffer = **new byte**[100];

**static** **void** IslemTamamlandi(IAsyncResult sonuc) {

FileStream f = (FileStream)sonuc.AsyncState;

**int** okunan = f.EndRead(sonuc);

f.Close();

}

AsyncCallback callBack = **new** AsyncCallback(IslemTamamlandi);

IAsyncResult sonuc = f.BeginRead(buffer, 0, buffer.Length, callBack, f);

VB.NET

**Shared** buffer(99) **As** **Byte**

**Shared** **Sub** IslemTamamlandi(**ByVal** sonuc **As** IAsyncResult)

**Dim** f **As** FileStream = **CType**(sonuc.AsyncState, FileStream)

**Dim** okunan **As** **Integer** = f.EndRead(sonuc)

f.Close()

**End** **Sub**

**Dim** f **As** FileStream = **New** FileStream("c:\xxx.dat", FileMode.Open)

**Dim** callBack **As** AsyncCallback = **New** AsyncCallback(**AddressOf** IslemTamamlandi)

**Dim** sonuc **As** IAsyncResult = f.BeginRead(buffer, 0, buffer.Length, \_  
 callBack, f)

Kod 269, Asenkron İşlem Tamamlandığında Haberdar Edilme Modeli

APM Metotları ve İstisna Kontrolü

BeginXXX metodu çağrıldığında bir istisna oluşsa dahi, ilgili istisna CLR tarafından fırlatılmamaktadır. Asenkron metotların kullanımında oluşabilecek istisnalar sadece EndXXX metotları çağrıldığında fırlatılmaktadır. Böylece, istisna kontrolünün sadece EndXXX metotları çağrıldığında yapılması yeterli olmaktadır.

Asenkron Delege Kullanımı

Asenkron delegeler, istenilen her metodun asenkron olarak kullanılabilmesini sağlamaktadır.

Aşağıdaki örnekte senkron olan FileStream.Read metodu ve asenkron delege kullanımı gösterilmiştir.

C#

// FileStream.Read metodu parametre listesine ve dönüş tipine uygun  
// olarak yeni delege tanımı yapılmaktadır.

**delegate** **int** FileRead(**byte** [] buffer, **int** offset, **int** count);

**public** **static** **void** Main() {

FileStream f = **new** FileStream(@"c:\xxx.dat", FileMode.Open);

**byte** [] buffer = **new** **byte**[100];

// fr isminde yeni bir delege tanımlanmakta, f.Read metodu bu delegeye  
 // atanmaktadır.

FileRead fr = f.Read;

IAsyncResult sonuc = fr.BeginInvoke(buffer, 0, buffer.Length, **null**, **null**);

// İşlemin tamamlanması beklenirken bu bölümde diğer işler yapılabilir.

fr.EndInvoke(sonuc);

}

VB.NET

' FileStream.Read metodu parametre listesine ve dönüş tipine uygun  
' olarak yeni delege tanımı yapılmaktadır.

**Delegate** **Function** FileRead(**ByVal** buffer **As** **Byte**(), **ByVal** offset **As** **Integer**,   
 **ByVal** count **As** **Integer**) **As** **Integer**

**Public** **Shared** **Sub** Main()

**Dim** f **As** FileStream = **New** FileStream("c:\xxx.dat", FileMode.Open)

**Dim** buffer(99) **As** **Byte**

' fr isminde yeni bir delege tanımlanmakta, f.Read metodu bu delegeye  
 ' atanmaktadır.

**Dim** fr **As** FileRead = **AddressOf** f.Read

**Dim** sonuc **As** IAsyncResult = fr.BeginInvoke(buffer, 0, buffer.Length, \_  
 **Nothing**, **Nothing**)

' İşlemin tamamlanması beklenirken bu bölümde diğer işler yapılabilir.

fr.EndInvoke(sonuc)

**End** **Sub**

Kod 270, Asenkron Delege Kullanım Örneği

BeginInvoke ve EndInvoke metotları kullanım özellikleri APM metotları olan BeginXXX ve EndXXX metotları ile aynıdır.

Asenkron delegeler, sağladıkları kullanım kolaylığı ile yeni iş parçacıkları oluşturarak yapılan çoklu iş parçacık uygulamalarına önemli bir alternatif sağlamaktadırlar.

BeginInvoke metodu, BeginXXX metodunda olduğu gibi ASyncCallback ve herhangi bir nesneyi parametre olarak alabilmektedir. Böylece istenilen metodun asenkron işlemin tamamlanması sonucunda otomatik olarak çağrılması sağlanmaktadır.

Aşağıdaki örnekte, şu ana kadar anlatılan iş parçacığı senkronizasyonu, istisna yönetimi ve asenkron delege kullanımı konularını örnekleyen kod parçası gösterilmiştir.

Kod parçası, System.Net.HttpWebRequest sınıfını kullanarak eş zamanlı olarak web sitesine istek göndermektedir. İsteğin başarıyla tamamlanması sonucu konsol ekranına web sitesinden dönen içerik yazılmaktadır.

C#

// Asenkron delege tanımı.

**delegate** WebResponse GetWebResponse();

**static** **object** sync = **new** **object**();

**static** **int** tamamlanan = 0;

**public** **static** **void** Main() {

HttpWebRequest req;

GetWebResponse resp;

AsyncCallback callBack;

// 10 adet asenkron çağrım yapılmakta ...

**for** (**int** i = 0; i < 10; i++) {

req = (HttpWebRequest)HttpWebRequest.Create(  
 "http://www.xxx.com/indir.aspxId=" + i.ToString());

resp = req.GetResponse;

callBack = **new** AsyncCallback(IslemTamamlandi);

// Parametre olarak işlem tamamlandığında çağrılması istenen  
 // metot ve IAsyncResult.AsyncState nesnesine atanmak istenen nesne  
 // gönderilmektedir.

resp.BeginInvoke(callBack, resp);

}

Console.WriteLine("İşlemin tamamlanması bekleniyor ...");

**lock** (sync) {

**while** (tamamlanan < 10) {

Thread.Sleep(10);

Monitor.Wait(sync);

}

}

Console.WriteLine("İşlem tamamlandı.");

}

**static** **void** IslemTamamlandi(IAsyncResult sonuc) {

Console.WriteLine("{0} numaralı iş parçacığı tamamlandı ...",   
 Thread.CurrentThread.ManagedThreadId);

// IAsyncResult.AsyncState ile delege elde edilmektedir.

GetWebResponse r = (GetWebResponse)sonuc.AsyncState;

**try** {

**using** (WebResponse resp = r.EndInvoke(sonuc)) {

**using** (StreamReader reader = **new** StreamReader(resp.GetResponseStream()))   
 {

Console.WriteLine(reader.ReadToEnd());

reader.Close();

}

resp.Close();

}

}

**catch** (Exception exc) {

Console.WriteLine(exc.Message);

}

**lock** (sync) {

tamamlanan++;

Monitor.Pulse(sync);

}

}

VB.NET

' Asenkron delege tanımı.

**Delegate** **Function** GetWebResponse() **As** WebResponse

**Shared** sync **As** **Object** = **New** **Object**()

**Shared** tamamlanan **As** **Integer** = 0

**Shared** **Sub** Main(**ByVal** args **As** **String**())

**Dim** req **As** HttpWebRequest

**Dim** resp **As** GetWebResponse

**Dim** callBack **As** AsyncCallback

' 10 adet asenkron çağrım yapılmakta ...

**For** i **As** **Integer** = 0 **To** 9

req = **CType**(HttpWebRequest.Create(("http://www.xxx.com/indir.aspx?Id=" + \_  
 i.ToString)), HttpWebRequest)

resp = **AddressOf** req.GetResponse

callBack = **New** AsyncCallback(**AddressOf** IslemTamamlandi)

' Parametre olarak işlem tamamlandığında çağrılması istenen  
 ' metot ve IAsyncResult.AsyncState nesnesine atanmak istenen nesne  
 ' gönderilmektedir.

resp.BeginInvoke(callBack, resp)

**Next**

Console.WriteLine("İşlemin tamamlanması bekleniyor ...")

**SyncLock** sync

**While** (tamamlanan < 10)

Thread.Sleep(10)

Monitor.Wait(sync)

**End** **While**

**End** **SyncLock**

Console.WriteLine("İşlem tamamlandı.")

**End** **Sub**

**Private** **Shared** **Sub** IslemTamamlandi(**ByVal** sonuc **As** IAsyncResult)

Console.WriteLine("{0} numaralı ii parçacığı tamamlandı ...", \_  
 Thread.CurrentThread.ManagedThreadId)

' IAsyncResult.AsyncState ile delege elde edilmektedir.

**Dim** r **As** GetWebResponse = **CType**(sonuc.AsyncState, GetWebResponse)

**Try**

**Using** resp **As** WebResponse = r.EndInvoke(sonuc)

**Using** reader **As** StreamReader = **New** StreamReader(resp.GetResponseStream)

Console.WriteLine(reader.ReadToEnd)

reader.Close()

**End** **Using**

resp.Close()

**End** **Using**

**Catch** exc **As** Exception

Console.WriteLine(exc.Message)

**End** **Try**

**SyncLock** sync

tamamlanan = tamamlanan + 1

Monitor.Pulse(sync)

**End** **SyncLock**

**End** **Sub**

Kod 271, İleri Seviye Senkronizasyon ve Asenkron Delege Kullanımı

Kod çıktısı aşağıda gösterilmiştir.

İşlemin tamamlanması bekleniyor ...

7 numaralı iş parçacığı tamamlandı ...

The remote server returned an error: (404) Not Found.

9 numaralı iş parçacığı tamamlandı ...

The remote server returned an error: (404) Not Found.

9 numaralı iş parçacığı tamamlandı ...

The remote server returned an error: (404) Not Found.

. . .

İşlem tamamlandı.

Listelenen kaynak kodlar \Bölüm 16\Delegeler klasöründe bulunmaktadır.

Küçük şeylere fazla önem verenler, ellerinden büyük şeyler gelmeyenlerdir.

Eflatun

Bölüm

17

Sık Kullanılan Sınıflar ve Öneriler

K

itabın bu bölümünde, .NET sınıf kütüphanesi içerisinde bulunan ve programlama sırasında sık kullanılan sınıflar hakkında bilgi ve kullanım önerileri sunulacaktır.

Okuyucu, bu bölümün tamamlanması ile;

* Tarih / Zaman İşlemleri
* Veri Formatlama
* Veri Dönüşümü
* Ortam Sınıfı Kullanımı
* Global Tekil Belirleyici Kullanımı
* Sistem Kayıt Defteri

Konularında bilgi sahibi olacaktır.

Tarih / Zaman İşlemleri

Programlama sırasında sık yapılan işlemlerden birisi de tarih - zaman değerleri üzerinde çalışmaktır.

System.DateTime yapı tipi, tarih / zaman bilgisini ifade eden tiptir. Zaman değerleri 100 nanosaniye aralıklar şeklinde tutulmaktadır. 100nano saniyelik zaman dilimine tik adı verilmektedir.

1 nanosaniye, saniyenin milyarda birine karşılık gelen değerdir.

.NET sınıf kütüphanesi içerisinde, zaman farkını standart bir yapıya kavuşturmak amacıyla System.TimeSpan yapı tipi tanımlanmıştır. System.Date yapı tipi tarih - zaman değerlerini yönetirken, System.TimeSpan yapı tipi iki tarih - zaman değeri arasındaki farklı yönetmek amacıyla kullanılmaktadır.

Tarih / Zaman Yapı Tipi

DateTime yapı tipi tarih ve saat değerlerini birlikte saklamakta ve yönetmektedir. Tarih / saat değerleri 01/01/0001 00:00:00 ile 12/31/9999 23:59:59 arasında herhangi bir değeri alabilmektedir.

Değerler, Gregorian takviminin 01/01/0001 00:00:00 tarihinden itibaren geçen tik değerleri ile ifade edilmektedir.

Aşağıda DateTime yapı tipinin önemli görülen üyeleri ve açıklamaları listelenmiştir.

| Üye Adı | Açıklama |
| --- | --- |
| Date | Tarih / zaman değerinin sadece tarih bölümünü döndürür. |
| Day | Değerin 1-31 arasında olan ay bazındaki gün bilgisini döndürür. |
| DayOfWeek | Tarihin haftanın hangi gününe denk geldiğini DayOfWeek sayılabilir tipi ile döndürür. |
| DayOfYear | Değerin 1-366 arasında olan yıl bazındaki gün bilgisini döndürür. |
| Hour | Değerin 0-23 arasında olan saat bilgisini döndürür. |
| Kind | Değer tarafından ifade edilen tarih / zaman bilgisinin yerel veya everensel olarak ifade edildiğini DateTimeKind sayılabilir tipi ile döndürür. |
| MaxValue | En büyük tarih / zaman değerini döndüren statik özelliktir. |
| MinValue | En küçük tarih / zaman değerini döndüren statik özelliktir. |
| Minute | Değerin 0-59 arasında olan dakika bilgisini döndürür. |
| Month | Değerin 1-12 arasında olan ay bilgisini döndürür. |
| Now | Bilgisayar saatine göre, yerel bazlı anlık tarih / saat değerini döndürür statik özelliktir. |
| Second | Değerin 0-59 arasında olan saniye bilgisini döndürür. |
| Ticks | Değeri oluşturan tik sayısını döndürür. |
| Today | Anlık tarih bilgisini döndürür statik özelliktir. |
| UtcNow | Anlık tarih / zaman değerini evrensel bazlı (UTC) olacak şekilde döndürür statik özelliktir. |
| Year | Değerin 0-9999 arasında olan yıl bilgisini döndürür. |
| Add | Parametre olarak iletilen zaman farkı nesnesini (TimeSpan) mevcut tarih / zaman değerine ekler. |
| AddXXXs | Parametre olarak iletilen değer kadar mevcut tarih / zaman değerine ekleme yapar.   Örneğin AddHours metodu belirtilen saati, AddMinutes değeri belirtilen dakika değerini ekler. |
| Compare | Parametre olarak iletilen iki adet tarih / zaman değerini karşılaştıran statik metottur. |
| DaysInMonth | Parametre olarak iletilen yıl ve ay değerlerine göre, ilgili ay içerisinde kaç gün olduğunu döndürür statik metottur. |
| FromBinary | Parametre olarak iletilen long değerinden tarih / zaman değeri döndüren statik metottur. Metot tarafından yapılan işlem DateTime değerinin de-serilize edilmesidir. |
| IsLeapYear | Parametre olarak iletilen yılın artık yıl olup olmadığını döndürür. |
| Subtract | Parametre olarak iletilen zaman farkı değerini mevcut tarih / zaman değerinden çıkartır. |
| ToBinary | Tarih / zaman değerini long olarak döndüren metottur. Metot tarafından yapılan işlem değerin serilize edilmesidir. |
| ToLongDateString | Tarih / zaman değerini uzun tarih formatında karakter dizi olarak döndüren metottur. Örneğin 08 Nisan 2007 Pazar gibi.   Not: Tüm formatlama işlemleri aktif iş parçacığı kültür bilgisine göre (Thread.CurrentThread.CurrentCulture) yapılmaktadır. |
| ToLongTimeString | Tarih / zaman değerini uzun zaman formatında karakter dizi olarak döndüren metottur. Örneğin 15:26:41 |
| TryParse | Parametre olarak iletilen karakter dizi değeri tarih / zaman değerine çevirebilmeyi kontrol eder, çevirme işlemi başarılı ile true değerini döndüren statik metottur. |

Tablo 99, System.DateTime Yapı Tipi Üye Bilgileri

Yapıcı Metot Bilgileri ve Kullanım Örnekleri

Tarih / zaman yapı tipi yüklenmiş farklı yapıcı metotları desteklemektedir. Aşağıdaki kod parçasında yapıcı metotlar ve çeşitli kullanım örnekleri gösterilmiştir.

C#

// Anlık tarih / zaman değeri.

DateTime t1 = DateTime.Now;

// tik sayısı ile tarih / zaman değeri oluşturulmaktadır.

DateTime t2 = **new** DateTime(999894449);

// Yıl, ay, gün değerlerinden tarih / zaman değeri oluşturulmaktadır.

DateTime t3 = **new** DateTime(2009,12,3);

// Yıl, ay, gün, saat, dakika, saniye değerlerinden tarih / zaman değeri

// oluşturulmaktadır.

DateTime t4 = **new** DateTime(2008, 7, 3, 22, 30, 0);

Console.WriteLine("t2: {0}", t2);

t3.AddMinutes(12);

t4.AddHours(2.5);

Console.WriteLine("t3: {0}", t3);

Console.WriteLine("t4 Tarih: {0}", t4.Date.ToLongDateString());

Console.WriteLine("t4 Zaman: {0}", t4.ToLongTimeString());

// 25 günlük zaman farklı değeri oluşturulmaktadır.

TimeSpan ts = TimeSpan.FromDays(25);

// Tarih / zaman yapı tipi operatör yüklemeyi desteklemektedir.

DateTime t5 = t2 + ts;

Console.WriteLine("t5: {0}", t5);

VB.NET

' Anlık tarih / zaman değeri.

**Dim** t1 **As** DateTime = DateTime.Now

' tik sayısı ile tarih / zaman değeri oluşturulmaktadır.

**Dim** t2 **As** DateTime = **New** DateTime(999894449)

' Yıl, ay, gün değerlerinden tarih / zaman değeri oluşturulmaktadır.

**Dim** t3 **As** DateTime = **New** DateTime(2009, 12, 3)

' Yıl, ay, gün, saat, dakika, saniye değerlerinden tarih / zaman değeri

' oluşturulmaktadır.

**Dim** t4 **As** DateTime = **New** DateTime(2008, 7, 3, 22, 30, 0)

Console.WriteLine("t2: {0}", t2)

t3.AddMinutes(12)

t4.AddHours(2.5)

Console.WriteLine("t3: {0}", t3)

Console.WriteLine("t4 Tarih: {0}", t4.Date.ToLongDateString())

Console.WriteLine("t4 Zaman: {0}", t4.ToLongTimeString())

' 25 günlük zaman farklı değeri oluşturulmaktadır.

**Dim** ts **As** TimeSpan = TimeSpan.FromDays(25)

' Tarih / zaman yapı tipi operatör yüklemeyi desteklemektedir.

**Dim** t5 **As** DateTime = t2 + ts

Console.WriteLine("t5: {0}", t5)

Kod 272, System.DateTime Yapı Tipi Kullanım Örnekleri

Kod çıktısı aşağıda gösterilmiştir.

t2: 01.01.0001 00:01:39

t3: 03.12.2009 00:00:00

t4 Tarih: 03 Temmuz 2008 Perşembe

t4 Zaman: 22:30:00

t5: 26.01.0001 00:01:39

Zaman Farkı Yapı Tipi

System.TimeSpan yapı tipi zaman farkını ifade etmektedir. Zaman farklı, tarih / zaman yapı tipinde olduğu gibi tik ile ifade edilmektedir.

Zaman farkının pozitif veya negatif (ileriye doğru, geriye doğru) olabileceğine dikkat edilmelidir.

Aşağıda System.TimeSpan yapı tipinin önemli görülen üyeleri ve açıklamaları listelenmiştir.

| Üye Adı | Açıklama |
| --- | --- |
| Days | Zaman farkı değeri içerisindeki toplam gün sayısını tamsayı olarak döndüren özelliktir. |
| Hours | Zaman farkı değeri içerisindeki toplam saat sayısını tamsayı olarak döndüren özelliktir. |
| MaxValue | En büyük zaman farkı değerini döndüren statik metottur. |
| MinValue | En küçük zaman farkı değerini döndüren statik metottur. |
| Minutes | Zaman farkı değeri içerisindeki toplam dakika sayısını tamsayı olarak döndüren özelliktir. |
| Seconds | Zaman farkı değeri içerisindeki toplam saniye sayısını tamsayı olarak döndüren özelliktir. |
| Ticks | Zaman farkı değeri içerisindeki tik gün sayısını döndüren özelliktir. |
| TotalDays | Zaman farkı değeri içerisindeki toplam gün sayısını ondalık değeri ile birlikte double olarak döndüren özelliktir. |
| TotalHours | Zaman farkı değeri içerisindeki toplam saat sayısını ondalık değeri ile birlikte double olarak döndüren özelliktir. |
| TotalMinutes | Zaman farkı değeri içerisindeki toplam dakika sayısını ondalık değeri ile birlikte double olarak döndüren özelliktir. |
| TotalSeconds | Zaman farkı değeri içerisindeki toplam saniye sayısını ondalık değeri ile birlikte double olarak döndüren özelliktir. |
| Zero | Sıfır zaman farkı değerini döndüren statik özelliktir. |
| Add | Parametre olarak iletilen zaman farkı değerini ekler. |
| Compare | Parametre olarak iletilen zaman farkı değeri ile karşılaştırma yapan statik metottur. |
| Duration | Mevcut zaman farkı mutlak değerini yeni bir zaman farkı yapı tipi olarak döndürür. |
| FromXXX | Parametre olarak iletilen değerden zaman farkı yapı tipi oluşturan statik metottur. |
| Negate | Mevcut zaman farkının negatif değerini yeni bir zaman farkı yapı tipi olarak döndürür. |
| Subtract | Parametre olarak iletilen zaman farkı değerini çıkarır. |

Tablo 100, System.TimeSpan Yapı Tipi Üye Bilgileri

Yapıcı Metot Bilgileri ve Kullanım Örnekleri

Zaman farkı yapı tipi yüklenmiş farklı yapıcı metotları desteklemektedir. Aşağıdaki kod parçasında yapıcı metotlar ve çeşitli kullanım örnekleri gösterilmiştir.

C#

TimeSpan ts1 = TimeSpan.MaxValue;

TimeSpan ts2 = **new** TimeSpan(6, 5, 45);

TimeSpan ts3 = TimeSpan.FromDays(30);

TimeSpan ts4 = ts2 - ts3;

TimeSpan ts5 = ts3 - ts2;

DateTime t = **new** DateTime(2009,12, 3);

Console.WriteLine("t+ts4: {0}", t + ts4);

Console.WriteLine("t+ts5: {0}", t + ts5);

TimeSpan ts6 = ts4.Duration();

Console.WriteLine("t+ts4.Duration: {0}", t + ts6);

ts6 = TimeSpan.FromMinutes(1.2);

Console.WriteLine("{0} bekleniyor ..", ts6);

Console.WriteLine("{0} sn bekleniyor",ts6.TotalSeconds);

Console.WriteLine("{0} st bekleniyor",ts6.TotalHours);

Thread.Sleep(TimeSpan.FromMinutes(1.2));

VB.NET

**Dim** ts1 **As** TimeSpan = TimeSpan.MaxValue

**Dim** ts2 **As** TimeSpan = **New** TimeSpan(6, 5, 45)

**Dim** ts3 **As** TimeSpan = TimeSpan.FromDays(30)

**Dim** ts4 **As** TimeSpan = ts2 - ts3

**Dim** ts5 **As** TimeSpan = ts3 - ts2

**Dim** t **As** DateTime = **New** DateTime(2009, 12, 3)

Console.WriteLine("t+ts4: {0}", t + ts4)

Console.WriteLine("t+ts5: {0}", t + ts5)

**Dim** ts6 **As** TimeSpan = ts4.Duration()

Console.WriteLine("t+ts4.Duration: {0}", t + ts6)

ts6 = TimeSpan.FromMinutes(1.2)

Console.WriteLine("{0} bekleniyor..", ts6)

Console.WriteLine("{0}sn bekleniyor",ts6.TotalSeconds)

Console.WriteLine("{0}st bekleniyor",ts6.TotalHours)

Thread.Sleep(TimeSpan.FromMinutes(1.2))

Kod 273, System.TimeSpan Yapı Tipi Kullanım Örnekleri

Kod çıktısı aşağıda gösterilmiştir.

t+ts4: 03.11.2009 06:05:45

t+ts5: 01.01.2010 17:54:15

t+ts4.Duration: 01.01.2010 17:54:15

00:01:12 bekleniyor ..

72 sn bekleniyor ..

0,02 saat bekleniyor ..

Veri Formatlama

Şu ana kadar kitapta verilen örneklerde veri formatlama kullanılmıştır. Örneğin Console.WriteLine metoduna format karakter dizi değeri ve parametreler iletilerek konsol penceresine doğru değerlerin yazılması sağlanmıştır.

Format Karakter Diziler

Format karakter dizi (ing. Format String), içerisinde { } karakterleri arasında format bilgisini içeren değerdir.

C#

**int** i = 1;

**double** d = 45.2;

**string** s = "selam";

**bool** b = **true**;

DayOfWeek w = DayOfWeek.Friday;

Console.WriteLine("i = {0}, d = {1}, s = {2}, b = {3}, w = {4}",

i, d, s, b, w);

VB.NET

**Dim** i **As** **Integer** = 1

**Dim** d **As** Double = 45.2

**Dim** s **As** **String** = "selam"

**Dim** b **As** **Boolean** = **True**

**Dim** w **As** DayOfWeek = DayOfWeek.Friday

Console.WriteLine("i = {0}, d = {1}, s = {2}, b = {3}, w = {4}", \_  
 i, d, s, b, w)

Kod 274, Örnek Format Karakter Dizi Kullanımı

Yukarıdaki örnekte farklı veri tiplerine sahip değerler Console.WriteLine metoduna parametre olarak iletilmiştir. İletilen parametre sırası ve format karakter dizi değeri sonucunda üretilen çıktı aşağıda gösterilmiştir.

i = 1, d = 45.2, s = selam, b = True, w = Friday

Burada yapılan işlem, format karakter dizi değeri içerisine özel karakterler ekleyerek veri formatlamanın sağlanmasıdır.

Veri formatlama amacıyla en sık kullanılan yöntemlerden birisi string.Format statik metodudur.   
  
Formatlama alt yapısının aktif kültür bilgisine göre çıktı ürettiğine dikkat edilmelidir.

{ } karakterleri arasında bulunan bilgiler, hangi parametre değerinin hangi pozisyona atanacağını belirtmektedir.

Format karakter dizi değeri üç bölümden oluşmaktadır. Her bölüm ; karakteri ile ayrılmaktadır.

{Bölüm1;Bölüm2;Bölüm3}

Bölüm2 ve Bölüm3 kullanımı isteğe bağlıdır. Aşağıda farklı bölümlerin birlikte kullanımı ve açıklamaları gösterilmiştir.

| Bölüm | Açıklama |
| --- | --- |
| Bölüm1 | Format karakter dizi değeri tüm parametrelere uygulanır. |
| Bölüm1;Bölüm2 | Bölüm1 sıfır ve pozitif değerler için, Bölüm2 negatif değerler için kullanılır. |
| Bölüm1;Bölüm2;Bölüm3 | Bölüm1 pozitif değerler için, Bölüm2 negatif değerler için, Bölüm3 sıfır değeri için kullanılır. |

Tablo 101, Format Karakter Dizi Bölümleri

Aşağıdaki örnekte format dizi bölümleri kullanımı gösterilmiştir.

| C# | VB.NET |
| --- | --- |
| **int** i = 1;  **int** j = -1;  **int** k = 0;  **string** f;  f = "{0:Pozitif;Negatif;Sıfır}";  Console.WriteLine(f, i);  Console.WriteLine(f, j);  Console.WriteLine(f, k); | **Dim** i **As** **Integer** = 1  **Dim** j **As** **Integer** = -1  **Dim** k **As** **Integer** = 0  **Dim** f **As** **String**  f = "{0:Pozitif;Negatif;Sıfır}"  Console.WriteLine(f, i)  Console.WriteLine(f, j)  Console.WriteLine(f, k) |

Kod 275, Format Karakter Dizi Bölümleri Kullanım Örneği

Kod çıktısı aşağıda gösterilmiştir.

Pozitif

Negatif

Sıfır

Belirtilen format dizi değerinin hatalı olması durumunda System.FormatException istisnası fırlatılacaktır.

Her bir bölüm aşağıdaki yapıda olmalıdır.

{İndeks [,Yaslama][:Format Değeri]}

İndeks, sıfırdan başlayan ve parametre sırasını belirten nümerik değerdir. İsteğe bağlı olan yaslama değerinin pozitif olması sağa yaslı, negatif olması sola yaslı formatlama yapılması gerektiğini belirtmektedir. Format değeri, farklı şekillerde veri formatlamayı sağlayan değerdir.

| C# | VB.NET |
| --- | --- |
| **int** i = 1;  **double** d = 43.63;  **string** s = "[Test]";  // 10 karakter sağa yaslı.  Console.WriteLine("{0,10}", i);  Console.WriteLine("{0,5:000.00}", i);  Console.WriteLine("{0,3:000.0}", d);  Console.WriteLine("{0,10}", s); | **Dim** i **As** **Integer** = 1  **Dim** d **As** Double = 43.63  **Dim** s **As** **String** = "[Test]"  ' 10 karakter sağa yaslı.  Console.WriteLine("{0,10}", i)  Console.WriteLine("{0,5:000.00}", i)  Console.WriteLine("{0,3:000.0}", d)  Console.WriteLine("{0,10}", s) |

Kod 276, Örnek Format Değerler ve Kullanımları

Kod çıktısı aşağıda gösterilmiştir. Sağa yaslamada formatlamanın boşluk (karakter 32) değerleri yapıldığına dikkat edilmelidir. Gösterimde boşluk karakteri → ile ifade edilmiştir.

→→→→→→→→→1

001,00

043,6

→→→→[Test]

Format Değerleri

Format değerleri farklı şekillerde veri formatlaması yapılmasını sağlamaktadır. Sırasıyla nümerik, tarih / zaman ve sayılabilir tipler için kullanılan format değerleri hakkında bilgiler sunulacaktır.

Kitapta, aktif kültür bilgisi tr-TR için örneklemeler yapılmıştır. Ondalık ayıracı, binlik ayıracı, para birimi gibi değerlerin farklı kültürlerde değişebileceğine dikkat edilmelidir.

Nümerik Veri Tipleri Format Değerleri

Aşağıda nümerik tipler için kullanılabilecek standart format değerleri gösterilmiştir.

| Değer | Açıklama |
| --- | --- |
| c | Değer parasal görünümde formatlanır. Örneğin {0:c} gibi. Format değeri hassasiyeti desteklemektedir. {0:c4} gibi.  Parasal sembol değeri ve yapısı aktif kültür bilgisine göre belirlenmektedir. |
| d | Tamsayı değerler için kullanılmaktadır. Format değeri hassasiyeti desteklemektedir. {0:d5} gibi. |
| e | Bilimsel format değeridir. Format değeri hassasiyeti desteklemektedir. {0:e5} gibi. |
| f | Sabit noktalı formatlama yapmaktadır. Format değeri hassasiyeti desteklemektedir. {0:d6} gibi. |
| g | Genel nümerik format değerdir. İletilen parametre tipi ve hassasiyet değerine göre en uygun şekilde formatlama yapılır. |
| n | Nümerik değer format değeridir. İletilen parametre değeri binlik ve ondalık ayıraçlarını içerecek şekilde formatlanır. Format değeri hassasiyeti desteklemektedir. {0:n4} gibi. |
| p | Yüzde değeri ifade etmektedir. Parametre olarak iletilen değeri 100 ile çarpar ve % sembolünü ekler. Format değeri hassasiyeti desteklemektedir. {0:p4} gibi. |
| r | Sadece double ve single tipleri için desteklenmektedir. Yuvarlanan değer sonucu oluşan karakter dizinin tekrar aynı değere dönüşebileceğini garanti etmektedir. |
| x | Onaltılık (ing. Hexadecimal) sisteme göre iletilen tamsayı değeri formatlar. x yerine X iletilmesi ondalık harfleri büyük olmasını sağlamaktadır. (ff yerine FF gibi) |

Tablo 102, Nümerik Standart Format Değerleri

Sunulan standart format değerlerine ek olarak istenilen şekilde formatlama yapabilmek amacıyla aşağıda listelenen format değerleri de desteklenmektedir.

| Değer | Açıklama |
| --- | --- |
| 0 | Bulunduğu pozisyonda değer yoksa 0 (sıfır), varsa ilgili rakamı çıktıya kopyalar. |
| # | Bulunduğu pozisyonda rakam varsa ilgili rakamı çıktıya kopyalar, rakam yoksa herhangi bir işlem yapılmaz. |
| . | Ondalık ayıracını belirtir. |
| , | Binlik ayıracını belirtir. |

Tablo 103, Nümerik Özel Format Değerleri

Nümerik format değerleri kullanım örnekleri aşağıda gösterilmiştir.

| Format Değeri | Sayı | Sonuç |
| --- | --- | --- |
| c4 | 120000 | 120.000,0000 TL |
| c | 120000 | 120.000,00 TL |
| d | 5400 | 5400 |
| d6 | 5400 | 005400 |
| e | 120000 | 1,200000e+005 |
| f | 120000 | 120000,00 |
| f | 43.123 | 43,12 |
| f4 | 43.123 | 43,1230 |
| g | 120000 | 120000 |
| g2 | 120000 | 1,2e+05 |
| n | 120000 | 120.000,00 |
| n | 43.123 | 43,12 |
| n | 43.127 | 43,13 |
| n4 | 43.127 | 43,1270 |
| p | 0.12 | %12,00 |
| x | 120000 | 1d4c0 |
| 000.00 | 12 | 012,00 |
| ###.## | 12 | 12 |
| #,# | 1200000 | 1.200.000 |
| ## ## ## | 123456 | 12 34 56 |

Tablo 104, Nümerik Format Değerleri Kullanım Örnekleri

Tarih / Zaman Format Değerleri

Aşağıda tarih / zaman değerleri için kullanılabilecek standart format değerleri gösterilmiştir.

| Değer | Açıklama |
| --- | --- |
| d | Kısa tarih formatı. |
| D | Uzun tarih formatı. |
| t | Kısa zaman formatı. |
| T | Uzun zaman formatı. |
| f | Kısa tarih zaman formatı. |
| F | Uzun tarih zaman formatı. |
| g | Kısa ön tanımlı tarih zaman formatı. |
| G | Uzun ön tanımlı tarih zaman formatı. |
| M | Ay formatı. |
| r | RFC1123 tarih zaman formatı. |
| s | Sıralanabilir tarih zaman formatı. |
| Y | Yıl formatı. |

Tablo 105, Tarih / Zaman Standart Format Değerleri

Sunulan standart format değerlerine ek olarak istenilen şekilde formatlama yapabilmek amacıyla aşağıda listelenen format değerleri de desteklenmektedir.

| Değer | Açıklama |
| --- | --- |
| dd | 1-31 arasında olan gün değeri. |
| ddd | Kısa gün adı değeri. |
| dddd | Gün adı değeri. |
| gg | Milattan önce / sonra değeri. |
| hh | Saat değeri. |
| HH | 24'lük saat değeri. |
| mm | Dakika değeri. |
| MM | 1-12 arasında olan ay değeri. |
| MMM | Kısa ay adı değeri. |
| MMMM | Ay adı değeri. |
| ss | 0-59 arasında olan saniye değeri. |
| tt | AM (Öğleden önce) ve PM (Öğleden sonra) değeri. |
| yy | İki haneli yıl değeri. |
| yyyy | Dört haneli yıl değeri. |
| zz | Zaman ofset değeri. |
| : | Zaman ayıraç değeri. |
| / | Tarih ayıraç değeri. |

Tablo 106, Tarih Zaman Özel Format Değerleri

Tarih / zaman format değerleri kullanım örnekleri aşağıda gösterilmiştir.

Örnekte 04/12/2009 13:00:07 tarih zaman değeri baz alınmıştır.

| Format Değeri | Sonuç |
| --- | --- |
| d | 04.12.2009 |
| D | 04.12.2009 |
| t | 13:00 |
| T | 13:00:07 |
| f | 04 Aralık 2009 Cuma 13:00 |
| M | 04 Aralık |
| s | 2009-12-04T13:00:07 |
| Y | Aralık 2009 |
| dd | 04 |
| MMM | Ara |
| hh:mm:ssss | 01:00:07 |
| dd/MM/yyyy | 04.12.2009 |

Tablo 107, Tarih / Zaman Format Değerleri Kullanım Örnekleri

Sayılabilir Tipler Format Değerleri

Aşağıda sayılabilir tipler için kullanılabilecek format değerleri gösterilmiştir.

| Değer | Açıklama |
| --- | --- |
| g | Ön tanımlı değer. |
| f | Üye isimleri. |
| d | Üye değerleri. |
| x | Onaltılık sistemde üye değerleri. |

Tablo 108, Sayılabilir Tipler Format Değerleri

Veri Dönüşümü ve Kullanım Bilgileri

Programlama sırasında sık yapılan işlemlerden birisi de farklı tipte olan verinin istenilen veriye dönüşümünün yapılmasıdır. Karakter dizi değer içersinde bulunan tarih / zaman değerinin tarih zaman yapı tipine veya mantıksal değerleri farklı veri tiplerine dönüştürme örnek buna olarak verilebilir.

Tip dönüşümü (ing. Type Casting) ile veri dönüşümü (ing. Data Conversion) kavramları birbirlerine karıştırılmamalıdır. Tip dönüşümünde hedeflenen amaç, aynı hafıza alanı içerisindeki veriye farklı bir tip ile referans etmek iken, veri dönüşümü, tip dönüşümü ile birlikte verinin işlenmesi ile bir başka veri tipinin oluşturulmasını esas almaktadır.

System.Convert sınıfı, veri dönüşümü için kullanılan temel sınıftır. Sınıf, sağlamış olduğu yüklenmiş statik üyeler yardımıyla bir çok verinin dönüşümünü yapabilmektedir. Dönüşüm işleminin başarısız olması durumunda System.FormatException istisnası fırlatılmaktadır.

Tip dönüşümünün başarısız olması durumunda System.InvalidCast istisnası fırlatıldığına dikkat edilmelidir.

Aşağıda Convert sınıfının önemli görülen üyeleri ve açıklamaları gösterilmiştir.

| Üye Adı | Açıklama |
| --- | --- |
| DbNull | Veri tabanı boş (null) değerine karşılık gelen özelliktir. |
| ChangeType | Parametre olarak iletilen nesneyi istenilen tipe dönüştürür. |
| FromBase64String | Parametre olarak iletilen Base64 kodlama standardında olan karakter dizi değerinden byte dizisi döndürür. |
| IsDbNull | Parametre olarak iletilen nesnenin DbNull değerine eşit olup olmadığını döndürür. |
| ToBase64String | Parametre olarak iletilen byte dizi değerinden Base64 kodlama standardına göre karakter dizi değeri döndürür. |
| ToXXX | Yüklenmiş sürümleri ile farklı verilerin dönüşümünü yapar. Örneğin ToDateTime, ToByte gibi. |

Tablo 109, System.Convert Sınıfı Üye Bilgileri

IConvertible Arabirimi

System.IConvertible arabirimi, veri tipinin farklı veri tiplerine dönüşebileceğini ifade eden arabirimdir.

Bir sınıf veya yapı tipinin bu arabirim gerçekleştirimini yapması ile ilgili tipin veri dönüşümünü desteklediği anlaşılmaktadır. Convert sınıfı metotları, desteklenen tipler için, veri dönüşümünde bu arabirim yardımıyla işlem yapmaktadırlar.

C#

**public** **interface** IConvertible {

TypeCode GetTypeCode();

**bool** ToBoolean(IFormatProvider provider);

**byte** ToByte(IFormatProvider provider);

**char** ToChar(IFormatProvider provider);

DateTime ToDateTime(IFormatProvider provider);

**decimal** ToDecimal(IFormatProvider provider);

**double** ToDouble(IFormatProvider provider);

. . .

}

VB.NET

**Public** **Interface** IConvertible

**Function** GetTypeCode() **As** TypeCode

**Function** ToBoolean(**ByVal** provider **As** IFormatProvider) **As** **Boolean**

**Function** ToByte(**ByVal** provider **As** IFormatProvider) **As** **Byte**

**Function** ToChar(**ByVal** provider **As** IFormatProvider) **As** **Char**

**Function** ToDateTime(**ByVal** provider **As** IFormatProvider) **As** DateTime

**Function** ToDecimal(**ByVal** provider **As** IFormatProvider) **As** **Decimal**

**Function** ToDouble(**ByVal** provider **As** IFormatProvider) **As** Double

. . .

**End** **Interface**

Kod 277, System.IConvertible Arabirim Tanımı

Aşağıda Convert sınıfının örnek üç metodunun kod gerçekleştirimi gösterilmiştir.

C#

// IConvertible arabirimi yardımıyla veri dönüşümü yapılmaktadır.

**public** **static** **bool** ToBoolean(DateTime value) {

**return** ((IConvertible)value).ToBoolean(**null**);

}

// Yapı tipinin metotları yardımıyla veri dönüşümü yapılmaktadır.

**public** **static** **decimal** ToDecimal(**string** value) {

**if** (value == **null**) {

**return** 0M;

}

**return** **decimal**.Parse(value, CultureInfo.CurrentCulture);

}

// Tip dönüşümü yardımıyla veri dönüşümü yapılmaktadır.

**public** **static** **decimal** ToDecimal(**double** value) {

**return** (**decimal**) value;

}

VB.NET

' IConvertible arabirimi yardımıyla veri dönüşümü yapılmaktadır.

**Public** **Shared** **Function** ToBoolean(**ByVal** value **As** DateTime) **As** **Boolean**

**Return** **DirectCast**(value, IConvertible).ToBoolean(**Nothing**)

**End** **Function**

' Yapı tipinin metotları yardımıyla veri dönüşümü yapılmaktadır.

**Public** **Shared** **Function** ToDecimal(**ByVal** value **As** **String**) **As** **Decimal**

**If** (value **Is** **Nothing**) **Then**

**Return** 0

**End** **If**

**Return** **Decimal**.Parse(value, CultureInfo.CurrentCulture)

**End** **Function**

' Tip dönüşümü yardımıyla veri dönüşümü yapılmaktadır.

**Public** **Shared** **Function** ToDecimal(**ByVal** value **As** Double) **As** **Decimal**

**Return** **CDec**(value)

**End** **Function**

Kod 278, System.Convert Sınıfı Örnek Metotları

Örneklerden de görüleceği üzere Convert sınıfı, farklı tiplerin dönüşümü için en uygun yöntemi uygulamaktadır.

Base64 Kodlama Standardı

Base64 kodlama standardı (ing. Base64 veya QuadroSexagesimal Encoding) pozisyon temelli, özellikle ikil verilerin kodlanması için kullanılan, sadece yazdırılabilir ASCII karakterleri üreten kodlama standardıdır.

Convert.FromBase64String ve Convert.ToBase64String metotları bu kodlama standardının uygulanabileceği metotlardır.

Aşağıdaki örnekte metotların kullanımı gösterilmiştir.

C#

**string** testDeger = "Test Değeri";

**byte** [] bytes = Encoding.UTF8.GetBytes(testDeger);

**string** kodlanmisDeger = Convert.ToBase64String(bytes);

Console.WriteLine("Base64 Kodlanmış Değer: {0}", kodlanmisDeger);

bytes = Convert.FromBase64String(kodlanmisDeger);

testDeger = Encoding.UTF8.GetString(bytes);

Console.WriteLine("Orjinal Değer: {0}", testDeger);

VB.NET

**Dim** testDeger **As** **String** = "Test Değeri"

**Dim** bytes() **As** **Byte** = Encoding.UTF8.GetBytes(testDeger)

**Dim** kodlanmisDeger **As** **String** = Convert.ToBase64String(bytes)

Console.WriteLine("Base64 Kodlanmış Değer: {0}", kodlanmisDeger)

bytes = Convert.FromBase64String(kodlanmisDeger)

testDeger = Encoding.UTF8.GetString(bytes)

Console.WriteLine("Orjinal Değer: {0}", testDeger)

Kod 279, Base64 Kodlama Örneği

Kod çıktısı aşağıda gösterilmiştir.

Base64 Kodlanmış Değer: VGVzdCBEZcSfZXJp

Orjinal Değer: Test Değeri

Base64FormattingOptions sayılabilir tipi, satır sonu ayıraçlarının çıktıya eklenip eklenmeyeceğini belirtmek için kullanılabilir.  
  
Base64 veri kodlama standardının herhangi bir şekilde veri güvenliği katmanı sağlamadığına dikkat edilmelidir.  
  
Veri kodlama sınıfları konusunda detaylı bilgi kitabın Girdi – Çıktı Yönetimi bölümünde sunulmuştur.

Ortam Sınıfı Kullanımı

System.Environment sınıfı, çalışılan ortam ve platform hakkında pratikte kullanılabilecek çeşitli bilgilere erişim için kullanılan sınıftır.

Aşağıda Environment sınıfı statik üyeleri ve açıklamaları gösterilmiştir.

| Üye Adı | Açıklama |
| --- | --- |
| CommandLine | Prosesin çalıştırılmasında kullanılan komut satır değerini döndürür. |
| CurrentDirectory | Aktif klasör değerini değiştirmek / elde etmek amacıyla kullanılmalıdır. |
| ExitCode | Proses çıkış değerini değiştirmek / elde etmek amacıyla kullanılmalıdır. |
| HasShutdownStarted | Windows işletim sistemi yeniden başlatma / kapatma işleminin başlaması durumunda true değerini döndürür. |
| MachineName | Bilgisayar adını döndürür. |
| NewLine | Aktif platform için geçerli yeni satır değerini döndürür.  Farklı platformlarda değeri değişebilen yeni satır değerini elde etmek için bu özelliğin kullanımı önerilmektedir. |
| OSVersion | İşletim sistemi sürüm bilgisini döndürür. |
| ProcessorCount | Sistemde bulunan işlemci sayısını döndürür. |
| TickCount | Windows işletim sisteminin başlamasından itibaren ne kadar süre geçtiğini tik değeri ile döndürür. |
| UserDomainName | Kullanıcı etki alanı bilgisini döndürür. |
| UserName | Prosesin çalıştırıldığı işletim sistemi kullanıcısı adını döndürür. |
| Version | .NET CLR sürüm bilgisini döndürür. |
| Exit | Prosesi kapatır ve parametre olarak iletilen çıkış değerini işletim sistemine iletir.  İletilen değerin sıfır olması proses tarafından yapılan işin başarıyla tamamlandığına, sıfırdan farklı olması hata oluştuğuna işaret etmektedir. |
| GetCommandLineArgs | Prosesi çalıştırmak için kullanılan komut satırı parametre değerini döndürür. |
| GetEnvironmentVariable | Parametre olarak iletilen ortam değişkeninin (ing. Environment Variable) değerini döndürür. |
| SetEnvironmentVariable | Parametre olarak iletilen ortam değişkeninin değerini değiştirir. |

Tablo 110, System.Environment Sınıfı Üye Bilgileri

Environment sınıfı kullanımının örneklendiği kod parçası aşağıda gösterilmiştir.

C#

Console.WriteLine(Environment.OSVersion.ToString());

Console.WriteLine(Environment.Version.ToString());

Console.WriteLine(Environment.GetEnvironmentVariable("PATH"));

VB.NET

Console.WriteLine(Environment.OSVersion.ToString())

Console.WriteLine(Environment.Version.ToString())

Console.WriteLine(Environment.GetEnvironmentVariable("PATH"))

Kod 280, System.Environment Sınıfı Kullanım Örnekleri

Kod çıktısı aşağıda gösterilmiştir.

Microsoft Windows NT 5.1.2600 Service Pack 2

2.0.50727.42

C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\ ...

Çıktı değerleri farklı işletim sistemi ve konfigürasyonlarda değişebilir.

Global Tekil Belirleyici Kullanımı

Global tekil belirleyiciler (ing. Globally Unique Identifiers – GUID), çakışma olasılıkları çok düşük olan 16Byte uzunluğundaki değerlerdir. Global tekil belirleyicilerin pratikte kullanım amacı tekil karakter dizi değerler elde etmektir.

System.Guid sınıfı global tekil belirleyici değerleri üretmek için kullanılmaktadır. Aşağıdaki örnekte bu sınıfın kullanımı gösterilmiştir.

C#

Guid id = Guid.NewGuid();

Console.WriteLine(id.ToString());

**foreach** (**byte** b **in** id.ToByteArray())

Console.Write("{0} ", b);

VB.NET

**Dim** id **As** Guid = Guid.NewGuid()

Console.WriteLine(id.ToString())

**For** **Each** b **As** **Byte** **In** id.ToByteArray()

Console.Write("{0} ", b)

**Next**

Kod 281, System.Guid Sınıfı Kullanım Örneği

Örnek kod çıktısı aşağıda gösterilmiştir.

843f1c3d-c38c-45b2-988e-eb2a87b1e9e0

61 28 63 132 140 195 178 69 152 142 235 42 135 177 233 224

Sistem Kayıt Defteri

Bu konu başlığında sistem kayıt defteri kullanımı hakkında özet bilgiler ve .NET sınıf kütüphanesi tarafından sağlanan sınıflar yardımıyla kayıt defteri üzerinde yapılabilecek işlemler hakkında bilgiler verilecektir.

Kayıt Defteri Hakkında

Sistem kayıt defteri (ing. System Registry), kullanıcı uygulama ayarlarından Windows işletim sisteminin kullandığı tanımlara kadar, istenilen bilgilerin saklanabileceği ve bilgiler üzerindeki güvenliğin sağlanabileceği veritabanı sağlamaktadır.

Kayıt Defteri Yapısı

Sistem kayıt defteri kullanım amacına göre beş farklı mantıksal gruba ayrılmıştır.

| Mantıksal Grup | Açıklama |
| --- | --- |
| HKEY\_CLASSES\_ROOT | Windows işletim sistemine yüklü uygulamalar ve dosya uzantıları hakkında bilgilerin saklandığı mantıksal gruptur. |
| HKEY\_CURRENT\_USER | İşletim sistemine giriş yapmış aktif kullanıcı ile ilgili tanımların saklandığı mantıksal bölümdür. |
| HKEY\_LOCAL\_MACHINE | Bilgisayar ve tüm sistem kullanıcılarını ilgilendiren ayarların saklandığı mantıksal bölümdür. |
| HKEY\_USERS | HKEY\_CURRENT\_USER grubunda bulunan anahtarlara karşılık gelen bilgilerin saklandığı mantıksal bölümdür. |
| HKEY\_CURRENT\_CONFIG | Anlık konfigürasyon hakkında bilgilerin saklandığı mantıksal bölümdür. |

Tablo 111, Sistem Kayıt Defteri Mantıksal Grupları

Kayıt defteri, mantıksal gruplar altında bulunan anahtarlar, anahtarlar içerisinde bulunan alt anahtarlar şeklinde organize edilmiştir. Her bir anahtar içerisinde ad, tip ve değer yapısında istenilen bilgilerin saklanabileceği alanlar bulunmaktadır.

Kayıt Defteri Veri Tipleri

Kayıt defteri farklı veri tiplerini desteklemektedir. Aşağıda desteklenen önemli veri tipleri ve açıklamaları listelenmiştir.

| Veri Tipi | Açıklama |
| --- | --- |
| REG\_BINARY | İkil veri tipi. |
| REG\_DWORD | Tamsayı veri tipi. |
| REG\_EXPAND\_SZ | Genişleyebilir karakter dizi değerleri için kullanılmaktadır. |
| REG\_SZ | Karakter veri tipi. |
| REG\_MULTI\_SZ | Boşluk veya virgül ile ayrılmış çoklu karakter dizi değerleri için kullanılmaktadır. |

Tablo 112, System.MulticastDelegate Sınıfı Üye Bilgileri

REG\_EXPAND\_SZ veri tipi ortam değişkenlerini içerebilmektedir. Bu veri tipine sahip bir değer okunurken değer açılmakta, ortam değişkeninin ifade ettiği değer döndürülmektedir. Örneğin %TEMP%, sistemde tanımlı geçici dosyaların tutulmasını belirten klasörü ifade etmektedir. Bu değerin REG\_EXPAND\_SZ tipinde olması durumunda ilgili klasör adı döndürülmektedir.

REG\_BINARY veri tipi ikil verilerin saklanması için kullanılmaktadır. Bu veri tipi ile serilize edilmiş nesneler byte dizisi olarak kayıt defterinde saklanabilmektedir.

Kullanım Bilgileri

Sistem kayıt defteri, Windows işletim sistemi ayarları ile sisteme giriş yapmış kullanıcı ile ilgili tanımların, uygulama ayarlarının ve istenilen her türlü saklanabilmesini sağlayan bir veritabanı sağlamaktadır.

Kayıt defteri içerisinde, teorik olarak, istenilen tip ve boyutta bilginin saklanması mümkündür. Buna rağmen yüksek boyutta verinin kayıt defteri içerisinde saklanması önerilmemektedir.

HKEY\_CURRENT\_USER mantıksal grubu içerisinde tutulan bilgiler, sadece işletim sistemine giriş yapan aktif kullanıcıya özgü olacak şekilde saklanmaktadır. Bu mantıksal grup içerisinde uygulama kullanıcı ayarlarının saklanması mümkündür. Böylece, işletim sistemine giriş yapan her kullanıcı için otomatik olarak farklı ayarların geçerli olması sağlanabilmektedir.

Örneğin HKEY\_CURRENT\_USER\ Software\Microsoft\Windows\CurrentVersion\Run isimli anahtar içerisinde, Windows işletim sistemine giriş yapan kullanıcı için otomatik olarak çalıştırılması istenen program listesi bulunmaktadır. Bu liste her bir kullanıcı bazında ayarlanabilmesine rağmen, HKEY\_LOCAL\_MACHINE\ Software\Microsoft\Windows\CurrentVersion\Run anahtarı, benzer işlevli listeyi, tüm kullanıcılar için tanımlamaktadır.

Bu kapsamda, uygulama ayar ve tanımlarının HKEY\_LOCAL\_MACHINE, kullanıcı bazlı ayar ve tanımların HKEY\_CURRENT\_USER mantıksal grubu içerisinde saklanması mümkündür.

Sistem kayıt defterinin, uygulama ve kullanıcı ayarlarının saklanması için kullanımı geçmişte Microsoft tarafından önerilmiş olsa da, özellikle uygulama dağıtımı ve sürüm yönetiminde getirmiş olduğu dezavantajlardan dolayı şu an için önerilmemektedir. Buna rağmen sistem kayıt defteri, doğru kullanıldığında, istenilen verinin güvenli şekilde saklanabilmesi için pratik bir veritabanı ortamı sağlamaktadır.

Veri tabanı uygulama yazılımlarında, veritabanı bağlantı dizesi (ing. Database Connection String) sistem kayıt defteri içerisinde saklanabilir.

Windows Kayıt Defteri Yöneticisi

Sistem kayıt defteri içerisinde bulunan bilgilerin yönetimi amacıyla Windows işletim sistemi tarafından Kayıt Defteri Yöneticisi uygulaması kullanıma sunulmuştur. Kayıt defteri yöneticisi regedit.exe komutu ile çalıştırılabilmektedir.

Ekran Çıktısı 52, Sistem Kayıt Defteri Yöneticisi

Kayıt defteri yöneticisi ile, sistem kayıt defteri üzerinde anahtar ekleme, silme ve istenilen değerlerin tanımının yapılabilmesi mümkündür.

Kayıt defteri yöneticisi ile kayıt defterinde saklanan tanımlar üzerinde yapılacak değişiklikler tüm sistemi etkileyebilmekte ve Windows işletim sisteminin doğru çalışmasını engelleyebilmektedir. Kayıt defteri içerisinde saklanan ve işletim sistemi ile ilgili olan ayarların Windows API (ing. Windows Application Proggramming Interface) fonksiyonları çağrılarak değiştirilmesi önerilmektedir.

Kayıt Defteri İşlemleri

Microsoft.Win32.Registry sınıfı, sistem kayıt defteri içerisindeki bilgilere erişim ve bilgiler üzerinde istenilen işlemlerin yapılabilmesini sağlayan metot ve özellikler içermektedir.

Bu bölümde, Registry sınıfı kullanılarak kayıt defteri üzerinde temel işlemlerin yapılabilmesi hakkında bilgiler sunulacaktır.

Tip Açıklamaları

Registry üye bilgi ve açıklamaları aşağıda listelenmiştir.

| Üye Adı | Açıklama |
| --- | --- |
| ClassesRoot | HKEY\_CLASSES\_ROOT mantıksal grubu içerisinde işlem yapabilmek için RegistryKey nesnesi döndüren statik özelliktir. |
| CurrentConfig | HKEY\_CURRENT\_CONFIG mantıksal grubu içerisinde işlem yapabilmek için RegistryKey nesnesi döndüren statik özelliktir. |
| LocalMachine | HKEY\_LOCAL\_MACHINE mantıksal grubu içerisinde işlem yapabilmek için RegistryKey nesnesi döndüren statik özelliktir. |
| Users | HKEY\_USERS mantıksal grubu içerisinde işlem yapabilmek için RegistryKey nesnesi döndüren statik özelliktir. |
| GetValue | Parametre olarak iletilen anahtar ve ad içerisindeki değeri döndüren statik metottur. Okuma HKEY\_CURRENT\_USER mantıksal grubu içerisinde yapılmaktadır. |
| SetValue | Parametre olarak iletilen anahtar ve ad içerisine değer yazmak için kullanılabilen statik metottur. Yazma HKEY\_CURRENT\_USER mantıksal grubu içerisinde yapılmaktadır. |

Tablo 113, Microsoft.Win32.Registry Sınıfı Üye Bilgileri

Registry.GetValue ve Registry.SetValue metotları, RegistryKey sınıfı kullanmadan sadece HKEY\_CURRENT\_USER mantıksal grubu içerisindeki verilere erişim için pratik bir yöntem sağlamaktadır.

Sistem kayıt defteri yönetimi için Microsoft.Win32 ad alanı içerisinde tanımlı yardımcı tipler ve açıklamaları aşağıda gösterilmiştir.

| Tip | Açıklama |
| --- | --- |
| RegistryKey | Kayıt defteri içerisindeki herhangi bir anahtarı simgeleyen sınıftır. |
| RegistryHive | Mantıksal grupları simgeleyen sayılabilir tiptir. |
| RegistryValueKind | Saklanan veri tipini simgeleyen sayılabilir tiptir. |

Tablo 114, Sistem Kayıt Defteri Yönetimi Yardımcı Tipleri

RegistryKey sınıfı IDisposable arabirim gerçekleştirimini yapmaktadır. Bu sınıfa ait nesneler ile yapılan işlemlerin using bloğu içerisinde yapılması önerilmektedir.

Kayıt Defteri Verilerine Erişim

Kayıt defteri içerisinde, mantıksal grup – anahtar ve anahtarların içerisinde ad, tip ve değer yapısında tutulan verilere erişim için öncelikle mantıksak grup içerisindeki ilgili anahtarın RegistryKey nesnesi ile elde edilmesi gerekmektedir.

Aşağıdaki kod parçasında, aktif kullanıcının işletim sistemine giriş yaptığında otomatik olarak çalıştırılan uygulama listesi kayıt defterinden okunmakta ve konsol ekranına yazdırılmaktadır.

C#

**string** anahtarYol = @"Software\Microsoft\Windows\CurrentVersion\Run";

// Registry.CurrentUser statik özelliği ile RegistryKey nesnesi elde   
// edilmektedir. RegistryKey.OpenSubKey metodu yardımıyla istenilen  
// anahtar açılmaktadır.

**using** (RegistryKey anahtar = Registry.CurrentUser.OpenSubKey(anahtarYol))

{

// RegistryKey.GetValueNames metodu, açılan anahtar içerisindeki değer  
 // adlarını döndürmektedir. RegistryKey.GetValue metodu ile istenilen ad  
 // değeri, RegistryKey.GetValueKind ile veri tipi elde edilmektedir.

**foreach** (**string** ad **in** anahtar.GetValueNames())

Console.WriteLine("{0}-{1}-{2}", ad, anahtar.GetValue(ad),   
 anahtar.GetValueKind(ad));

**a**nahtar.Close()

};

VB.NET

**Dim** anahtarYol **As** **String** = "Software\Microsoft\Windows\CurrentVersion\Run"

' Registry.CurrentUser statik özelliği ile RegistryKey nesnesi elde   
' edilmektedir. RegistryKey.OpenSubKey metodu yardımıyla istenilen  
' anahtar açılmaktadır.

**Using** anahtar **As** RegistryKey = Registry.CurrentUser.OpenSubKey(anahtarYol)

' RegistryKey.GetValueNames metodu, açılan anahtar içerisindeki değer  
 ' adlarını döndürmektedir. RegistryKey.GetValue metodu ile istenilen ad  
 ' değeri, RegistryKey.GetValueKind ile veri tipi elde edilmektedir.

**For** **Each** ad **As** **String** **In** anahtar.GetValueNames

Console.WriteLine("{0}-{1}-{2}", ad, anahtar.GetValue(ad), \_

anahtar.GetValueKind(ad))

**Next**

anahtar.Close()

**End** **Using**

Kod 282, Kayıt Defteri Verilerine Erişim Örneği

Veri Yazma

Aşağıdaki kod parçasında, anahtarın kayıt defteri içerisinde oluşturulması ve oluşturulan anahtar içerisine değer yazma gösterilmiştir.

C#

**string** anahtarYol = @"Software\Firma\Uygulama";

// RegistryKey.CreateSubKey metodu ile istenilen anahtar oluşturulmaktadır.

**using** (RegistryKey anahtar = Registry.LocalMachine.CreateSubKey(anahtarYol)) {

// RegistryKey.SetValue metodu ile istenilen ad, değer ve veri tipi  
 // kayıt defterine yazılmaktadır.

anahtar.SetValue("Lisans", "xyz");

anahtar.SetValue("Bilgi", 123456, RegistryValueKind.DWord);

anahtar.SetValue("OrnekIkil", **new** **byte**[] { 45, 34, 54 },

RegistryValueKind.Binary);

// Genişleyebilir karakter dizi veri tipi ile %ProgramFiles% isimli ortam  
 // değeri yazılmaktadır. Bu değerin okunmasında çıkan sonuca dikkat   
 // ediniz.

anahtar.SetValue("OrnekStr", @"%ProgramFiles%\Uygulama",   
 RegistryValueKind.ExpandString);

**foreach** (**string** ad **in** anahtar.GetValueNames())

Console.WriteLine("{0}-{1}-{2}", ad, anahtar.GetValue(ad),   
 anahtar.GetValueKind(ad));

anahtar.Close();

};

// RegistryKey.DeleteSubKeyTree metodu ile istenilen anahtar, tüm alt  
// anahtarları ile birlikte silinmektedir.

Registry.LocalMachine.DeleteSubKeyTree(@"Software\Firma");

VB.NET

**Dim** anahtarYol **As** **String** = "Software\Firma\Uygulama"

' RegistryKey.CreateSubKey metodu ile istenilen anahtar oluşturulmaktadır.

**Using** anahtar **As** RegistryKey = Registry.LocalMachine.CreateSubKey(anahtarYol)

' RegistryKey.SetValue metodu ile istenilen ad, değer ve veri tipi  
 ' kayıt defterine yazılmaktadır.

anahtar.SetValue("Lisans", "xyz")

anahtar.SetValue("Bilgi", 123456, RegistryValueKind.DWord)

anahtar.SetValue("OrnekIkil", **New** **Byte**() {45, 34, 54}, \_  
 RegistryValueKind.Binary)

' Genişleyebilir karakter dizi veri tipi ile %ProgramFiles% isimli ortam  
 ' değeri yazılmaktadır. Bu değerin okunmasında çıkan sonuca dikkat   
 ' ediniz.

anahtar.SetValue("OrnekStr", "%ProgramFiles%\Uygulama", \_  
 RegistryValueKind.ExpandString)

**For** **Each** ad **As** **String** **In** anahtar.GetValueNames()

Console.WriteLine("{0}-{1}-{2}", ad, anahtar.GetValue(ad), \_

anahtar.GetValueKind(ad))

**Next**

anahtar.Close()

**End** **Using**

Registry.LocalMachine.DeleteSubKeyTree("Software\Firma")

Kod 283, Sistem Kayıt Defteri Veri Düzenleme Örneği

Kod çıktısı aşağıda gösterilmiştir.

Lisans-xyz-String

Bilgi-123456-DWord

OrnekIkil-System.Byte[]-Binary

OrnekStr-C:\Program Files\Uygulama-ExpandString

RegistryKey sınıfı ile kayıt defteri üzerinde yapılan değişiklikler performansın artırılması amacıyla anında kayıt defterine yansıtılmamakta, RegistryKey.Close metodunun çağrılması ile kayıt defterine fiziksel yazma yapılmaktadır. RegistryKey.Flush metodu, değişikliklerin kayıt defterine fiziksel olarak yazılmasını sağlamak için kullanılmalıdır.

İlim ve sanat iltifat görmediği ülkeleri terk eder.

İbn-i Sina

Bölüm

18

Olay ve Performans Yönetimi

W

indows işletim sistemi tarafından sağlanan işlevlerin uygulama yazılımlarında kullanılıyor olması, zaman ve emekten tasarruf edebilmeyi beraberinde getirmektedir.

Bu bölümde Windows tarafından sağlanan önemli iki işlev ve kullanımları konularında bilgiler sunulacaktır.

Sistem Olay Yönetimi

Windows işletim sistemi tarafından, gerek işletim sisteminin gerek uygulama yazılımlarının ürettiği olay bilgilerinin merkezi olarak toplanabilmesini sağlamak amacıyla olay günlüğü (ing. Event Log) hizmeti sunulmaktadır.

Genel Kavram ve Araçlar

Bu konu başlığında, olay yönetimi kapsamında kullanılan hizmet ve uygulama yazılımları ile olay kaynakları hakkında bilgiler verilecektir.

Olay Günlüğü Hizmeti

Olay günlüğü hizmeti, işletim sisteminin aktif olması ile birlikte arka planda bir hizmet uygulaması (ing. Service Application) olarak çalışmaya başlamaktadır.

Ekran Çıktısı 53, Sistem Olay Yönetici Hizmeti

Olay günlüğü hizmetinin işlevi, farklı işletim sistemi parçaları ve uygulama yazılımlarından gelen isteklere yanıt vermek ve olayların merkezi olarak yönetimini yapmaktır.

Olay Görüntüleyicisi

Olay görüntüleyicisi (ing. Event Viewer), olay günlüğü hizmeti tarafından yazılan olay bilgilerini görüntüleme, olay günlüğünü temizleme veya olayları dışa aktarmak (ing. Export) amacıyla kullanılan Microsoft yönetim konsol uygulamasıdır (ing. Microsoft Management Console Application).

Ekran Çıktısı 54, Sistem Olay Görüntüleyicisi

Olay Günlükleri, Tipleri ve Kaynakları

Ön tanımlı üç temel olay günlüğü kullanıma sunulmuştur. Bunlar uygulama (ing. Application), güvenlik (ing. Security) ve sistem (ing. System) olay günlükleridir. Olay yönetim sistemi, temel olay günlüklerine ek olarak yeni olay günlükleri eklenebilmesini de desteklenmektedir. Mevcut olay günlükleri olay görüntüleyicisinde ağaç yapısında görülmektedir.

Olay tipi (ing. Event Type), olay günlüğüne eklenen olayların bilgi, uyarı veya hata niteliğinde olduğunu belirten kavramdır.

Olay kaynağı (ing. Event Source), olayı ekleyen işletim sistemi parçası veya uygulama yazılımını tanımlayan karakter dizi değerdir.

Olay Günlüğü Kullanımı

Sistem olay günlüğü, uygulama yazılımlarının yönetici kullanıcıya iletmek istediği uyarı, hata veya bilgi amaçlı mesajları saklamak ve yönetmek için uygun bir zemin sağlamaktadır.

Olay günlüğünün pratikte en sık kullanıldığı işlev, yazılımda oluşan istisnaların olay günlüğüne tüm detaylarıyla yazılmasıdır. Tek noktadan yönetilebilme, dışa aktarılabilme gibi sağlamış olduğu avantajlar ile olay günlüğü sistemi, istisnaların saklanması, bilgi ve uyarı amaçlı her mesajın yönetici kullanıcıya iletimi amacıyla uygulama yazılımları tarafından kullanılabilir.

Olay günlüğü kullanımında, günlüğe yazılan olay boyutunun toplam 64KB'ı geçmemesine özen gösterilmesidir. Özellikle Windows Vista işletim sisteminde olay boyutu ile ilgili bazı kısıtlamalar getirilmiştir.

Kullanımda dikkat edilmesi gereken konulardan birisi de güvenliktir. Olay günlüğüne bilgi eklenebilmesi için Windows işletim sistemi kullanıcısının yeterli ayrıcalıklara sahip olması gerekmektedir.

Olay günlük tanımları HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Services\Eventlog kayıt defteri anahtarı içerisinde saklanmaktadır. İstenilen olay günlüğüne, istenilen işlem için yetki vermek amacıyla sistem kayıt defteri yöneticisi yazılımı kullanılabilir. Bunun için ilgili anahtar üzerinde sağ tıklayıp "Permissions …" komutu vermek yeterlidir.

.NET Olay Günlük Sınıfı

System.Diagnostics.EventLog sınıfı olay günlüğü işlemlerinin kolaylıkla yapılabilmesini sağlayan sınıftır. Bu bölümde en sık kullanılan işlevlerin bu sınıf ile yapılması hakkında bilgi verilecektir.

Olay Günlüğü Bilgilerine Erişim

Olay günlüğü bilgilerine erişimi örnekleyen kod parçası aşağıda gösterilmiştir.

C#

**using** (EventLog log = **new** EventLog("System")) {

**foreach** (EventLogEntry e **in** log.Entries) {

Console.WriteLine("{0}, {1}, {2}, {3} {4}",

e.EntryType, e.TimeGenerated, e.Message,

e.Category, e.UserName, e.Source);

}

log.Close();

}

VB.NET

**Using** log **As** EventLog = **New** EventLog("System")

**For** **Each** e **As** EventLogEntry **In** log.Entries

Console.WriteLine("{0}, {1}, {2}, {3}", e.EntryType, e.TimeGenerated,\_  
 e.Message, e.Category, e.UserName, e.Source)

**Next**

log.Close()

**End** **Using**

Kod 284, Sistem Olay Günlüğü Bilgilerine Erişim

Kod parçasında log nesnesi System isimli olay günlüğünü yönetmek amacıyla oluşturulmuştur. EventLog.Entries özelliği ile erişilen günlük bilgilerinin sırasıyla olay tipi, olayın oluştuğu tarih, mesajı, kategorisi, kullanıcı adı ve olay kaynağı konsol ekranına yazdırılmaktadır.

Olay Kaynağı Oluşturma

Olay günlüğüne eklenen olayların hangi kaynaktan geldiğini belirtmek amacıyla kullanılan olay kaynağı, EventLog.CreateEventSource statik metodu ile oluşturulmaktadır.

Olay günlüğüne ekleme yaparken, olayın hangi kaynaktan geldiği belirtilmelidir.

Aşağıdaki kod parçasında yeni bir olay kaynağı oluşturulması gösterilmiştir.

C#

// TestUygulama isminde olay kaynağı yoksa Application olay günlüğü  
// içerisinde oluştur.

**if** (!EventLog.SourceExists("TestUygulama"))

EventLog.CreateEventSource("TestUygulama", "Application");

// Oluşturulan kaynağı sil.

EventLog.DeleteEventSource("TestUygulama");

VB.NET

' TestUygulama isminde olay kaynağı yoksa Application olay günlüğü  
' içerisinde oluştur.

**If** **Not** EventLog.SourceExists("TestUygulama") **Then**

EventLog.CreateEventSource("TestUygulama", "Application")

**End** **If**

' Oluşturulan kaynağı sil.

EventLog.DeleteEventSource("TestUygulama");

Kod 285, Olay Kaynağı Oluşturma Örneği

Aşağıdaki kod parçasında yeni olay günlüğü oluşturulması ve olay günlüklerinin listelenmesi gösterilmiştir.

C#

// Projeler isminde olay günlüğü yok ise oluştur.

**if** (!EventLog.Exists("Projeler"))

EventLog.CreateEventSource("TestUygulama", "Projeler");

// Her bir olay günlüğü için günlük adı ve günlük dosya limitini yazdır.

**foreach** (EventLog log **in** EventLog.GetEventLogs()) {

Console.WriteLine("{0}, {1}, {3}", log.Log, log.LogDisplayName,   
 log.MaximumKilobytes);

}

' Olay günlüğünü sil.

EventLog.Delete("Projeler");

VB.NET

' Projeler isminde olay günlüğü yok ise oluştur.

**If** **Not** EventLog.Exists("Projeler") **Then**

EventLog.CreateEventSource("TestUygulama", "Projeler")

**End** **If**

' Her bir olay günlüğü için günlük adı ve günlük dosya limitini yazdır.

**For** **Each** log **As** EventLog **In** EventLog.GetEventLogs

Console.WriteLine("{0}, {1}, {2}", log.Log, log.LogDisplayName, \_  
 log.MaximumKilobytes)

**Next**

' Olay günlüğünü sil.

EventLog.Delete("Projeler")

Kod 286, Yeni Olay Günlüğü Oluşturulması

Kod çıktısı aşağıda gösterilmiştir.

Application, Application, 1536

Internet Explorer, Internet Explorer, 512

Projeler, Projeler, 512

Security, Security, 512

System, System, 512

Olay Günlüğüne Yazma

Olay günlüğüne bilgi yazmak amacıyla EventLog.WriteEntry metodu kullanılmalıdır. WriteEntry metodu yüklenmiş sürümleri, yazılan olayın kategori ve tip bilgilerini de desteklemektedir.

Örnek kod parçasında, farklı tiplerde olay ekleme işlemi gösterilmiştir.

C#

**if** (!EventLog.Exists("Projeler"))

EventLog.CreateEventSource("OrnekUygulama", "Projeler");

EventLog log = **new** EventLog("Projeler");

log.Source = "OrnekUygulama";

log.WriteEntry("Test mesaj ...", EventLogEntryType.Information);

log.WriteEntry("Hata mesajı ...", EventLogEntryType.Error);

log.WriteEntry("Uyarı mesajı ...", EventLogEntryType.Warning);

VB.NET

**If** **Not** EventLog.Exists("Projeler") **Then**

EventLog.CreateEventSource("OrnekUygulama", "Projeler")

**End** **If**

**Dim** log **As** EventLog = **New** EventLog("Projeler")

log.Source = "OrnekUygulama"

log.WriteEntry("Test mesaj ...", EventLogEntryType.Information)

log.WriteEntry("Hata mesajı ...", EventLogEntryType.**Error**)

log.WriteEntry("Uyarı mesajı ...", EventLogEntryType.Warning)

Kod 287, Olay Günlüğüne Yazma Örneği

Kod parçasının çalıştırılması ardından olay görüntüleyicisi ekran çıktısı aşağıda gösterilmiştir.

Ekran Çıktısı 55, Olay Günlüğüne Eklenen Bilgiler

Performans Sayaçları

Windows işletim sistemi tarafından sunulan performans sayaçları, işletim sistemi ve uygulama yazılımlarının performans analizinin yapılabilmesi ve izlenebilmesi için alt yapı sağlamaktadır.

Uygulama yazılımlarında performansın istenilen düzeyde olmaması durumunda, hangi işlem veya işlemlerde olası problemler yaşandığı performans sayaçları kullanılarak analiz edilebilmektedir.

Performans sayaçlarının sadece performans problemlerinin analizinde kullanılabileceği düşünülmemelidir. Gerek tek, gerek çoklu iş parçacık uygulamalarında performans sayaçları, uygulama yazılımı tarafından yapılan işlem sayıları, ortalama süreleri gibi değerleri elde etmek amacıyla da kullanılabilmektedir.

Uygulama performans analizi ve performans sayaçlarının tüm özellikleriyle kullanımı oldukça detaylı bir konudur. Kitapta okuyucuya bu konuda teorik düzeyde temel bilgiler ve pratikte en sık kullanılan yöntemler hakkında bilgiler sunulacaktır.

Genel Kavram ve Araçlar

Genel mimari, performans kategorileri (performans nesneleri) ve her bir kategori altında bulunan performans sayaçları şeklinde organize edilmiştir.

Kategoriler, birbirleriyle ilişkili performans sayaçlarını belirli bir grup altında toplamak amacıyla kullanılmaktadır. Örneğin işlemci kategorisi altında işlemcilerin kullanım ve boş kalma oranları gibi işlemciler ile ilgili farklı performans sayaçları bulunmaktadır.

Performans Günlük Uygulaması

Performans günlük uygulaması (ing. Performance Logs and Alerts), Windows işletim sistemi ile birlikte yüklenen performans analiz aracıdır.

Performans günlük uygulaması ile sistemde bulunan performans sayaç değerleri liste veya grafiksel olarak görüntülenebilmekte, performans sayaçları değerleri izlenebilmektedir.

Performans günlük uygulamasının kontrol panel | yönetimsel araçlar klasöründe bulunan kısa yol veya Microsoft yönetim konsolu aracılığı ile çalıştırılması mümkündür.

Microsoft yönetim konsolunu çalıştırmak için mmc.exe komutu verilmelidir. File | Add / Remove Snapin … komutu ile performans günlük uygulaması seçilerek uygulama başlatılabilir.

Aşağıda performans günlük uygulaması ana penceresi gösterilmiştir.

Ekran Çıktısı 56, Performans Günlük Uygulaması

Ön tanımlı görünüm olan çizgisel grafik görüntüleme, Ctrl-R veya View Report komutu verilerek rapor görünümüne dönüştürülebilir. Rapor görünümü sayaç değerlerinin daha net izlenebilmesini sağlamaktadır.

Ekran Çıktısı 57, Performans Günlük Uygulaması Rapor Bazlı Görünüm

Yukarıdaki ekran çıktısında, MYLAPTOP isimli bilgisayar üzerinde bulunan Memory, PhysicalDisk ve Processor isimli performans kategorileri görüntülenmektedir. Her bir performans kategorisi içerisinde bulunan performans sayaç değerleri ekran çıktısında gözükmektedir.

CTRL-I veya rapor görünümü üzerinde sağ tıklayarak açılan menüden Add Counters… komutu verilerek sistemde bulunan farklı performans sayaç değerlerinin görüntülenmesi mümkündür.

Aşağıda ASP.NET web uygulamaları performans analizinde sıklıkla kullanılan bazı performans sayaçlarının eklenmesi gösterilmiştir.

Ekran Çıktısı 58, Yeni Performans Sayaçları Eklenmesi

Ekran çıktısından da görüleceği üzere, ASP.NET Apps … isimli performans kategorisi altında bir çok performans sayacı bulunmaktadır.

Ekran Çıktısı 59, Örnek Performans Sayaçları

Örneğin Requests/Sec isimli performans sayacı web uygulamasına bir saniyede ortalama kaç istek geldiğini belirtmektedir. Forms Authentication Failure isimli performans sayacı ile, form kimlik doğrulama yöntemi ile kaç isteğin kimlik doğrulama adımından geçemediği ifade edilmektedir.

Tekil ve Çoklu Örnek Sayaç Kategorileri

Performans sayaç kategorileri tekil örnekli (ing. Single Instance) veya çoklu örnekli (ing. Multiple Instance) niteliğinde olabilmektedir.

Örneğin bilgisayarda bulunan işlemci performans analizinin yapılabilmesi için Processor isimli performans kategorisi ve bu kategori altında çeşitli performans sayaçları bulunmaktadır. İşlemcinin ne kadar boşta kaldığı oranını %Idle Time isimli sayaç değeri ile elde edilebilmektedir.

Bilgisayarda bir işlemci bulunması durumunda tek sayaç bir işlemcinin boşta kalma oranını elde etmek için yeterlidir. Buna rağmen, sistemde birden fazla işlemci bulunduğu durumlarda aynı bilginin her bir işlemci için görüntülenmesi gerekebilir. Sayaç tek olmasına rağmen, sayaç ile ifade edilen bilginin bu durumda her bir işlemci için tekrarlanması, bir başka anlatımla sayacın çoklu örnekli olması gerekmektedir.

Ekran Çıktısı 60, Çoklu Sayaç Kategorisi Örneği

Ekran çıktısında, %Idle Time isimli sayaç gösterilmiştir. Sağ bölümdeki örnek listesi (Instance List) içerisinde bulunan eleman sayısı, sistemde bulunan işlemci sayısına göre artmaktadır.

Tekil örnek niteliğinde olan sayaçlar, yukarıda anlatılan gereksinimin bulunmadığı durumlarda kullanılabilir. Örneğin Memory kategorisi buna örnek olarak verilebilir.

Ekran Çıktısı 61, Tekil Sayaç Kategorisi Örneği

Sayaç Tipleri

Performans sayaçları, sayaç kategorisi, sayacı belirten sayaç adı, sayaç hakkında detaylı bilgi sağlayan yardım ve sayaç tipi değerlerinden oluşmaktadır.

Sayaç tipi, sayacın bir sonraki değerinin hesaplanacağı formülü ifade etmektedir. Bazı sayaç değerleri yüzde ve oranlar ile ifade edilirken, bazı sayaç değerleri için gerçekleşme sayıları anlamlı olmaktadır.

Aşağıda bilgisayar hafızası kategorisi içerisinde bulunan iki sayaç bilgisi gösterilmiştir.

Ekran Çıktısı 62, Performans Sayaç Tip Örnekleri

Available MBytes isimli sayaç, hafızada kullanılabilir durumda kaç MB alan bulunduğunu ifade ederken, Page Reads/Sec isimli sayaç saniyede hafızada kaç sayfa okuması yapıldığını ifade etmektedir. İlk sayaç için toplam miktar değeri anlamlı iken (NumberOfItems64), ikinci sayaç için 1 saniyelik süre içerisinde yapılan işlem sayısı (RateOfCountsPerSecond32) anlamlıdır.

Aşağıdaki tabloda önemli görülen performans sayaç tip ve açıklamaları gösterilmiştir.

| Sayaç Tipi | Açıklama |
| --- | --- |
| AverageTimer32 | Bir işlemin ortalama tamamlanma süresini belirten sayaç tipidir. |
| CounterDelta32 | Son iki ölçümde oluşan farklı belirten sayaç tipidir. |
| ElapsedTime | İşlemin başlaması ile sayaç değerinin hesaplanması arasında geçen süreyi belirten sayaç tipidir. |
| NumerOfItems32 | Sayaç değerini herhangi bir hesaplama yapmadan belirten sayaç tipidir. |
| RateOfCountsPerSecond | Bir saniye içerisinde ortalama yapılan işlem sayısını belirten sayaç tipidir. |

Tablo 115, Performans Sayaç Tip ve Açıklamaları

Performans Sayaçları Oluşturma

Uygulama yazılımlarında istenen işlemlerin performans analizi Yeni performans sayaç ve kategorileri oluşturularak kolaylıkla yapılabilir.

Performans sayaç ve kategorilerinin oluşturulması ile performans sayaçlarının değerlerinin değiştirilmesi farklı işlemlerdir. Kitabın ilerleyen konularında performans sayaç değerleri ile ilgili bilgi sunulacaktır.

Performans sayaçları VS.NET ile veya .NET sınıfı kütüphanesi sınıfları ile oluşturulabilir.

VS.NET ile Performans Sayaçları Yönetimi

VS.NET ile performans sayaç ve kategorileri oluşturmak için, sunucu gezgini (ing. Server Explorer) penceresi içerisinden istenilen bilgisayara ait performans sayaçları elemanı seçilmelidir.

Ekran Çıktısı 63, VS.NET Sunucu Gezgini Penceresi

Elemanın genişletilmesi ile sistemde bulunan performans kategorileri görüntülenecektir. Yeni bir performans kategorisi eklemek için Performance Counters elemanı üzerinde sağ tıklanarak New Category komutu verilmelidir. Açılan pencerede performans kategorisi ve performans sayaçları tanımlanabilmektedir.

Mevcut kategorinin ve kategori içerisindeki performans sayaçlarının düzenlenmesi için ilgili kategori ismi üzerinde sağ tıklanarak açılan menüden Edit Category komutu kullanılmalıdır.

Ekran Çıktısı 64, VS.NET ile Performans Sayaç Yönetimi

Ekran çıktısında yeni bir kategori ve kategori içerisinde OrnekSayac isimli, NumberOfItems32 tipinde yeni bir performans sayacı eklenmesi gösterilmektedir. Eklenen kategori ve sayaç performans değerlerinin performans günlük uygulaması ile izlenmesi mümkündür.

Ekran Çıktısı 65, VS.NET ile Oluşturulan Performans Sayacı

.NET Sınıfları ile Performans Sayaçları Yönetimi

System.Diagnostics.PerformanceCounterCategory ve System.Diagnostics.CounterCreationData sınıfları performans sayaçları ve kategorilerinin oluşturulması için kullanılan sınıflardır.

Aşağıdaki kod parçasında yeni performans kategorisi ve kategori içerisinde farklı tipte performans sayaçlarının oluşturulması gösterilmiştir.

C#

// Ornek Kategori isminde performans kategorisi var ise öncelikle mevcut  
// kategoriyi sil.

**if** (PerformanceCounterCategory.Exists("Ornek Kategori"))

PerformanceCounterCategory.Delete("Ornek Kategori");

// İsim, yardım ve tip bilgileri ile performans sayaçlarını oluşturma  
// bilgisini oluştur.

CounterCreationData pc1 = **new** CounterCreationData(  
 "Ortalama Yapılan İşlem Sayısı", "", PerformanceCounterType.SampleCounter);

CounterCreationData pc2 = **new** CounterCreationData(  
 "Toplam Yapılan İşlem Sayısı", "", PerformanceCounterType.NumberOfItems32);

// Ortalama hesaplayan performans sayaç tipi için, sayaç değeri ve sayaç   
// değerinin hesaplanmasını sağlayan yardımcı sayaç eklenmektedir.

CounterCreationData pc3 = **new** CounterCreationData(  
 "Ortalama İşlem Süresi", "", PerformanceCounterType.AverageTimer32);

CounterCreationData pc4 = new CounterCreationData(  
 "Ortalama İşlem Süresi Temel Sayaç","", PerformanceCounterType.AverageBase);

// Oluşturma bilgilerini koleksiyona ekle.

CounterCreationDataCollection coll = **new** CounterCreationDataCollection();

coll.Add(pc1);

coll.Add(pc2);

coll.Add(pc3);

coll.Add(pc4);

// Performans kategorisi ve kategori içerisindeki performans sayaçlarını  
// oluştur.

PerformanceCounterCategory.Create("Ornek Kategori", "",   
 PerformanceCounterCategoryType.MultiInstance, coll);

VB.NET

' Ornek Kategori isminde performans kategorisi var ise öncelikle mevcut  
' kategoriyi sil.

**If** PerformanceCounterCategory.Exists("Ornek Kategori") **Then**

PerformanceCounterCategory.Delete("Ornek Kategori")

**End** **If**

' İsim, yardım ve tip bilgileri ile performans sayaçlarını oluşturma  
' bilgisini oluştur.

**Dim** pc1 **As** CounterCreationData = **New** CounterCreationData(  
 "Ortalama Yapılan İşlem Sayısı", "", PerformanceCounterType.SampleCounter)

**Dim** pc2 **As** CounterCreationData = **New** CounterCreationData(  
 "Toplam Yapılan İşlem Sayısı", "", PerformanceCounterType.NumberOfItems32)

' Ortalama hesaplayan performans sayaç tipi için, sayaç değeri ve sayaç   
' değerinin hesaplanmasını sağlayan yardımcı sayaç eklenmektedir.

**Dim** pc3 **As** CounterCreationData = **New** CounterCreationData(  
 "Ortalama İşlem Süresi", "", PerformanceCounterType.AverageTimer32)

**Dim** pc4 **As** CounterCreationData = **new** CounterCreationData(  
 "Ortalama İşlem Süresi Temel Sayaç", "", PerformanceCounterType.AverageBase)

' Oluşturma bilgilerini koleksiyona ekle.

**Dim** coll **As** CounterCreationDataCollection = \_  
 **New** CounterCreationDataCollection()

coll.Add(pc1)

coll.Add(pc2)

coll.Add(pc3)

' Performans kategorisi ve kategori içerisindeki performans sayaçlarını  
' oluştur.

PerformanceCounterCategory.Create("Ornek Kategori", "", \_  
 PerformanceCounterCategoryType.MultiInstance, coll)

Kod 288, .NET Sınıfları ile Performans Sayaçları Oluşturma

Kod parçasında kategorinin çoklu örnekli niteliğinde oluşturulduğuna dikkat edilmelidir. PerformanceCounterCategoryType.SingleInstance sayılabilir tip değeri tekil örnekli performans kategorisi oluşturmak için kullanılmalıdır.

Kod parçasının çalıştırılması ile oluşturulan kategori ve sayaçlar aşağıda gösterilmiştir.

Ekran Çıktısı 66, .NET Sınıfları ile Oluşturulan Performans Sayaçları

Performans Sayaç Sınıfı

System.Diagnostics.PerformanceCounter sınıfı, performans sayaç değerleri üzerinde işlem yapmak amacıyla kullanılmaktadır.

C#

**public** **sealed** **class** PerformanceCounter : Component, ISupportInitialize

VB.NET

**Public** **NotInheritable** **Class** PerformanceCounter

**Inherits** Component

**Implements** ISupportInitialize

Kod 289, System.Diagnostics.PerformanceCounter Sınıf Tanımı

Yapıcı Metot Bilgileri

PerformaceCounter sınıfı yüklenmiş yapıcı metotları, kategori ve sayaç adı ile sayacın salt okunur olup olması ve çoklu sayaç kategorileri için örnek adını (ing. Instance Name) parametre olarak almaktadır.

Aşağıda önemli görülen yüklenmiş yapıcı metot tanım ve açıklamaları gösterilmiştir.

C#

// Parametre almayan yapıcı metottur.

**public** PerformanceCounter();

// Kategori ve sayaç adı ile sayacın salt okunurluk özelliğinin  
// iletildiği yapıcı metottur.

**public** PerformanceCounter(**string** categoryName, **string** counterName,   
 **bool** readOnly);

// Sadece çoklu örnekli sayaçlarda, kategori ve sayaç adı ile birlikte  
// örnek adı ve salt okunurluk özelliğinin iletildiği metottur.

**public** PerformanceCounter(**string** categoryName, **string** counterName,   
 **string** instanceName, **bool** readOnly);

VB.NET

' Parametre almayan yapıcı metottur.

**Public** **Sub** **New**()

' Kategori ve sayaç adı ile sayacın salt okunurluk özelliğinin  
' iletildiği yapıcı metottur.

**Public** **Sub** **New**(**ByVal** categoryName **As** **String**, **ByVal** counterName **As** **String**,\_

**ByVal** [**readOnly**] **As** **Boolean**)

' Sadece çoklu örnekli sayaçlarda, kategori ve sayaç adı ile birlikte  
' örnek adı ve salt okunurluk özelliğinin iletildiği metottur.

**Public** **Sub** **New**(**ByVal** categoryName **As** **String**, **ByVal** counterName **As** **String**,\_  
 **ByVal** instanceName **As** **String**, **ByVal** [**readOnly**] **As** **Boolean**)

Kod 290, PerformanceCounter Sınıfı Yapıcı Metot Tanımları

Üye Bilgileri

PerformanceCounter sınıfı üye bilgi ve açıklamaları aşağıda listelenmiştir.

| Üye Adı | Açıklama |
| --- | --- |
| CategoryName | Sayacın ait olduğu kategori adını değiştirmek / elde etmek amacıyla kullanılmalıdır. |
| CounterName | Sayaç adını değiştirmek / elde etmek amacıyla kullanılmalıdır. |
| CounterType | Sayaç tipini değiştirmek / elde etmek amacıyla kullanılmalıdır. |
| InstanceName | Çoklu örnekli sayaçlar için örnek adını değiştirmek / elde etmek amacıyla kullanılmalıdır. |
| ReadOnly | Sayacın salt okunurluk özelliğini değiştirmek / elde etmek amacıyla kullanılmalıdır. |
| RawValue | Sayacın hesaplanmamış anlık değerini döndürür özelliktir. |
| Decrement | Sayaç değerini bir azaltır. |
| Increment | Sayaç değerini bir artırır. |
| IncrementBy | Sayaç değerini parametre olarak iletilen değer kadar artırır. |
| NextValue | Performans sayacının hesaplanan bir sonraki değerini döndürür. |
| RemoveInstance | Çoklu örnekli sayaçlar için örneği kaldırır. |

Tablo 116, System.Diagnostics.PerformanceCounter Sınıfı Üye Bilgileri

Örnek Senaryo

Performans sayaçları ile ilgili aktarılan teorik bilgilerin pratikte kullanımı ile ilgili senaryo aşağıdadır.

Programlama sırasında sık yapılan işlemlerden birisi de döngü içerisinde karakter dizi değerlerin birleştirilmesidir.   
  
Bilindiği gibi bu işlemin karakter dizi değerlerin + operatörü ile birleştirilerek yapılması önerilmemekte, bunun yerine System.Text.StringBuilder sınıfı kullanımı önerilmektedir.  
  
Örnek senaryo, performans sayaçları kullanılarak her iki yöntemin performans analizinin yapılmasını içermektedir. Geliştirilecek analiz uygulaması içerisinde iki iş parçacığı oluşturulmalı ve her iki yöntemin performans sonuçları paralel olarak izlenebilmelidir.

Genel Tasarım ve Sınıflar

PerformansTest isimli sınıf, performans sayaç ve kategorilerinin oluşturulması ve her iki yöntemin performans analizini yapmak amacıyla kullanılmaktadır.

Yapılan test PerformansTest.Basla metodunun farklı iş parçacıkları tarafından çalıştırılması ile başlamaktadır.

C#

// Test tipini belirten sayılabilir tiptir.

**enum** TestTipi { StringTest, StringBuilderTest }

**class** PerformansTest { . . . }

**public** **static** **void** Main() {

// İki adet test nesnesi oluştur.

PerformansTest stringTest = **new** PerformansTest(TestTipi.StringTest);

PerformansTest sbTest = **new** PerformansTest(TestTipi.StringBuilderTest);

// İki adet iş parçacığı oluştur.

Thread stringThread = **new** Thread(stringTest.Basla);

Thread sbThread = **new** Thread(sbTest.Basla);

// İş parçacıklarına isim ver. Verilen isimler sayaç örnekleri olarak  
 // kullanılmaktadır.

stringThread.Name = "String";

sbThread.Name = "StringBuilder";

// İş parçacıklarını başlat.

stringThread.Start();

sbThread.Start();

Console.WriteLine("İşlemin tamamlanması bekleniyor ...");

stringThread.Join();

sbThread.Join();

}

VB.NET

' Test tipini belirten sayılabilir tiptir.

**Enum** TestTipi

StringTest

StringBuilderTest

**End** **Enum**

**Class** PerformansTest

**End** **Class**

**Public** **Shared** **Sub** Main()

' İki adet test nesnesi oluştur.

**Dim** stringTest **As** PerformansTest = **New** PerformansTest(TestTipi.StringTest)

**Dim** sbTest **As** PerformansTest

sbTest = **New** PerformansTest(TestTipi.StringBuilderTest)

' İki adet iş parçacığı oluştur.

**Dim** stringThread **As** Thread = **New** Thread(**AddressOf** stringTest.Basla)

**Dim** sbThread **As** Thread = **New** Thread(**AddressOf** sbTest.Basla)

' İş parçacıklarına isim ver. Verilen isimler sayaç örnekleri olarak

' kullanılmaktadır.

stringThread.Name = "String"

sbThread.Name = "StringBuilder"

' İş parçacıklarını başlat.

stringThread.Start()

sbThread.Start()

Console.WriteLine("İşlemin tamamlanması bekleniyor ...")

stringThread.Join()

sbThread.Join()

**End** **Sub**

Kod 291, Performans Testinin Başlatılması

Performans Sayaçlarının Oluşturulması

PerformansTest sınıfı statik yapıcı metodu içerisinde, sistemde performans sayaçlarının bulunmaması durumunda uygulama çalıştığında sayaçlar oluşturulmaktadır.

C#

**class** PerformansTest {

**private** **const** **string** KategoriAd = "String Performans Testleri";

**private** **const** **string** OrtalamaIslemSayisiAd = "İşlem Sayısı / sn";

**private** **const** **string** ToplamIslemSayisiAd = "Toplam İşlem Sayısı";

**private** **const** **string** OrtalamaIslemSuresiAd = "Ortalama İşlem Süresi";

**private** **static** **void** Olustur() {

**if** (PerformanceCounterCategory.Exists(KategoriAd))

PerformanceCounterCategory.Delete(KategoriAd);

CounterCreationData pc1 = **new** CounterCreationData(OrtalamaIslemSayisiAd,   
 "", PerformanceCounterType.RateOfCountsPerSecond32);

. . .

}

**static** PerformansTest() {

**if** (!PerformanceCounterCategory.Exists(KategoriAd))

Olustur();

}

. . .

}

VB.NET

**Class** PerformansTest

**Private** **Const** KategoriAd **As** **String** = "**String** Performans Testleri"

**Private** **Const** OrtalamaIslemSayisiAd **As** **String** = "İşlem Sayısı / sn"

**Private** **Const** ToplamIslemSayisiAd **As** **String** = "Toplam işlem Sayısı"

**Private** **Const** OrtalamaIslemSuresiAd **As** **String** = "Ortalama İşlem Süresi"

**Private** **Shared** **Sub** Olustur()

**If** PerformanceCounterCategory.Exists(KategoriAd) **Then**

PerformanceCounterCategory.Delete(KategoriAd)

**End** **If**

**Dim** pc1 **As** CounterCreationData

pc1 = **New** CounterCreationData(OrtalamaIslemSayisiAd, "", \_  
 PerformanceCounterType.RateOfCountsPerSecond32)

. . .

**End** **Sub**

**Shared** **Sub** **New**()

**If** **Not** PerformanceCounterCategory.Exists(KategoriAd) **Then**

Olustur()

**End** **If**

**End** **Sub**

. .

**End** **Class**

Kod 292, Performans Sayaçlarının Oluşturulması

Sayaç Değerlerinin Artırılması

Yapılan her bir işlemin ardından tanımlı sayaç değerleri artırılmaktadır. Sayaç değerlerini artırmak amacıyla PerformansTest.Artir metodu kullanılmaktadır.

C#

**class** PerformansTest {

**private** PerformanceCounter ortalamaIslemSayisiSayaci;

**private** PerformanceCounter toplamIslemSayisiSayaci;

**private** PerformanceCounter ortalamaIslemSuresiSayaci;

**private** PerformanceCounter ortalamaIslemSuresiTemelSayaci;

**private** **void** SayacNesneleriOlustur() {

ortalamaIslemSayisiSayaci = **new** PerformanceCounter(  
 KategoriAd, OrtalamaIslemSayisiAd, Thread.CurrentThread.Name, **false**);

toplamIslemSayisiSayaci = **new** PerformanceCounter(  
 KategoriAd, ToplamIslemSayisiAd, Thread.CurrentThread.Name, **false**);

. . .

}

**private** **void** Artir(**long** tik) {

ortalamaIslemSayisiSayaci.Increment();

toplamIslemSayisiSayaci.Increment();

ortalamaIslemSuresiSayaci.IncrementBy(tik);

ortalamaIslemSuresiTemelSayaci.Increment();

}

. . .

}

VB.NET

**Class** PerformansTest

**Private** ortalamaIslemSayisiSayaci **As** PerformanceCounter

**Private** toplamIslemSayisiSayaci **As** PerformanceCounter

**Private** ortalamaIslemSuresiSayaci **As** PerformanceCounter

**Private** ortalamaIslemSuresiTemelSayaci **As** PerformanceCounter

**Private** **Sub** SayacNesneleriOlustur()

ortalamaIslemSayisiSayaci = **New** PerformanceCounter(KategoriAd, \_  
 OrtalamaIslemSayisiAd, Thread.CurrentThread.Name, **False**)

. . .

**End** **Sub**

**Private** **Sub** Artir(**ByVal** tik **As** **Long**)

ortalamaIslemSayisiSayaci.Increment()

toplamIslemSayisiSayaci.Increment()

ortalamaIslemSuresiSayaci.IncrementBy(tik)

ortalamaIslemSuresiTemelSayaci.Increment()

**End** **Sub**

. . .

**End** **Class**

Kod 293, Sayaç Değerlerinin Artırılması

Performans Test Metodu

İş parçacıkları tarafından çağrılan PerformansTest.Basla metodu içeriği aşağıda gösterilmiştir.

C#

**public** **void** Basla() {

SayacNesneleriOlustur();

StringBuilder sb = **new** StringBuilder();

**string** sonuc = **string**.Empty;

**long** tikSayisi;

**for** (**int** i = 0; i < 300000; i++) {

tikSayisi = DateTime.Now.Ticks;

**for** (**int** j = 0; j < 1000; j++) {

// tip değerine göre işlemde StringBuilder sınıfı kullan.

**if** (tip == TestTipi.StringBuilderTest)

sb.Append("01234");

**else**

sonuc = sonuc + "01234";

}

// İşlem süresini parametre olarak ilet ve sayaç değerlerini artır.

Artir(DateTime.Now.Ticks - tikSayisi);

}

}

VB.NET

**Public** **Sub** Basla()

SayacNesneleriOlustur()

**Dim** sb **As** StringBuilder = **New** StringBuilder()

**Dim** sonuc **As** **String** = **String**.Empty

**Dim** tikSayisi **As** **Long**

**For** i **As** **Integer** = 1 **To** 300000

tikSayisi = DateTime.Now.Ticks

**For** j **As** **Integer** = 1 **To** 1000

' tip değerine göre işlemde StringBuilder sınıfı kullan.

**If** tip = TestTipi.StringBuilderTest **Then**

sb.Append("01234")

**Else**

sonuc = sonuc + "01234"

**End** **If**

**Next**

' İşlem süresini parametre olarak ilet ve sayaç değerlerini artır.

Artir((DateTime.Now.Ticks - tikSayisi))

**Next**

**End** **Sub**

Kod 294, Performans Test Metodu

Performans test metodunda, "01234" karakter dizi değeri StringBuilder.Append metodu veya karakter dizi birleştirme operatörü kullanılarak eklenmektedir. 1000 kez yapılan bu işlem ardından performans sayaçları değerleri bir artırılmaktadır. Test toplam 300.000 kez tekrarlanmaktadır.

Performans Değerlerinin İzlenmesi

Test başladıktan yaklaşık 20 sn sonra performans günlük uygulaması ile elde edilen anlık sonuçlar aşağıda gösterilmiştir.

Ekran Çıktısı 67, Performans Test Sonuçları

Ekran çıktısındaki bilgilerden, StringBuilder sınıfı kullanılarak yapılan işlemin yaklaşık 30 kat daha hızlı olduğu sonucu çıkmaktadır.

Listelenen kaynak kodlar \Bölüm 18\Senaryo klasöründe bulunmaktadır.

Bir yerde küçük insanların büyük gölgeleri oluşuyorsa o yerde güneş batıyor demektir.

Çin Atasözü

Bölüm

19

Veri Güvenliği ve Elektronik İmza

İ

nternet tarafından sağlanan olanakların pratikte kullanımının yaygınlaşması ve açık sistemler, veri güvenliği kavramının daha da önem kazanmasını beraberinde getirmiştir. Bu bölümde, çok geniş bir konu olan veri güvenliği, veri şifreleme algoritmaları, özet bilgi ve elektronik imza kavramları çerçevesinde aktarılacaktır.

Temel Kavramlar

Kriptografi (ing. Cryptography), veri şifreleme konusunu inceleyen bilim dalıdır. Kriptografi ve şifrelenmiş verinin analizini kapsayan kriptoanaliz (ing. Cryptanalysis) ile birlikte kriptoloji (ing. Cryptology), matematiğin şifre bilim dalına verilen isimdir.

Veri güvenliği aşağıda listelenen üç önemli konuyu içermektedir.

* Gizlilik, verinin istenmeyen kişiler tarafından okunabilirliğini engelleme.
* Bütünlük, verinin herhangi bir değişikliğe uğramadığının belirlenebilmesi.
* İnkar edememe**,** veriyi ileten tarafın, gönderdiği veriyi inkar edememesi.

Gizlilik, verinin şifrelenebilmesini (ing. Encryption) ve şifrelenmiş verinin orijinal veriye dönüştürülebilmesini, yani çözülebilmesini (ing. Decryption) ifade etmektedir. Orijinal veri düz metin (ing. Plain Text), şifrelenmiş veri şifreli metin (ing. Chipher Text) olarak tanımlanmaktadır. Veri şifrelemenin yapılabilmesi amacıyla geliştirilmiş farklı şifreleme algoritmaları bulunmaktadır.

Bütünlük, iletilen veya saklanan verinin orijinal sürümü ile aynı olduğunun kanıtlanabilmesini belirtmektedir. Bütünlüğün sağlanabilmesi için farklı yöntemler kullanılmaktadır.

İnkar edememe, veriyi oluşturan kişinin, veriyi oluşturduğu haliyle inkar edememesini ifade etmektedir. İnkar edememe, elektronik imza kullanımı ile sağlanan işlevlerdendir.

Veri Şifreleme Yöntem ve Algoritmaları

Veri şifreleme, düz metni kaynak olarak alıp, farklı şifreleme algoritmaları ile verinin anlaşılamaz şekilde şifreli metne dönüştürülmesidir. Şifreleme işleminde en önemli üç konu, kullanılan şifreleme anahtarı (ing. Encryption Key), şifreleme algoritması ve şifre çözme anahtarıdır (ing. Decryption Key).

Şekil 40, Veri Şifreleme ve Çözme İşlemi

Veri şifreleme ve şifre çözme anahtarı kavramı ile anlatılmak istenen belirli uzunluğa sahip veridir.

Veri şifreleme algoritmaları, verinin şifrelenmesi ve çözülmesinde farklı matematiksel formüller kullanmaktadır. Şifreleme algoritmaları, sağladığı güvenlik seviyesi, şifreleme / şifre çözme performansı ve kullandığı yöntemlere göre çeşitlilik göstermektedir.

Günümüzde kabul görmüş ve pratikte kullanılan belirli veri şifreleme algoritmaları bulunmaktadır. Veri şifreleme konusunda yapılabilecek en büyük hatalardan birisi, standart olarak benimsenmiş veri şifreleme algoritmalarını kullanmak yerine, her programcının kendi veri şifreleme algoritmasını geliştirmeye çalışmasıdır. Kabul görmüş şifreleme algoritmaları ve algoritmaların gerçekleştirimleri uzun emek ve deneyim sonucu ortaya çıkmıştır. Şifreleme konusunda seçilen algoritmadan daha kritik olan şifreleme ve şifre çözme işleminde kullanılan anahtardır.

Şifreleme algoritmaları ve anahtarları kapı kilitlerine benzetilebilir. Nasıl ki kapı kilitlerinin çalışma mekanizması aynı olmasına rağmen, kapıları açmak için kullanılan anahtarlar farklı ise, veri şifreleme işleminde de aynı senaryo geçerlidir. Veri şifrelemede herkes tarafından aynı veya birkaç farklı algoritmanın kullanılıyor olmasının veri güvenlik düzeyine herhangi bir olumsuz etkisi bulunmamaktadır.

Veri şifreleme güvenlik düzeyi, şifrelemede seçilen anahtar uzunluğu ile değişmektedir. Seçilen anahtar boyutunun büyük olması veri şifreleme güvenlik düzeyini artıran en önemli etkendir. Buna rağmen, seçilen anahtarın uzun olmasının şifreleme ve şifre çözme performansına olumsuz etkileri olabilmektedir.

Simetrik Şifreleme ve Algoritmaları

Simetrik şifreleme algoritması (ing. Symmetric Encryption Algorithm), veri şifreleme ve çözme işleminde kullanılan anahtarın aynı olduğu algoritmalara verilen isimdir.

Şekil 41, Simetrik Şifreleme İşlemi

Simetrik şifreleme algoritmaları yüksek miktarda veriyi, seçilen anahtar uzunluğu ve veri boyutuna bağlı olarak, çok hızlı şifreleyebilmekte ve çözebilmektedir.

Günümüzde en çok kullanılan simetrik şifreleme algoritmaları, destekledikleri anahtar boyutları ve açıklamaları aşağıda gösterilmiştir.

| Algoritma | Anahtar Boyutu (Bit) | Açıklama |
| --- | --- | --- |
| Rijndael, AES | 128, 192, 256 | AES, Advanced Encryption Standart, bir çok devlet tarafından şifreleme standardı olarak benimsenmiş simetrik şifreleme algoritmasıdır. Algoritma Rijndael algoritması olarak da bilinmektedir. |
| DES | 56 | DES, Data Encrption Standart, uzun yıllar kullanılmış olmasına rağmen, yeterli güvenliği sağlayabilen anahtar boyutunu desteklememektedir. |
| 3DES | 112 | 3DES, Triple DES, DES ile yapılan işlemin üç kez tekrarlanarak daha uzun anahtarı destekleyen şifreleme algoritmasıdır. |

Tablo 117, Simetrik Şifreleme Algoritmaları ve Açıklamaları

Veri şifreleme amacıyla simetrik algoritma kullanılması durumunda, veriyi şifreleyen taraf ile şifreyi çözen tarafın aynı anahtar değerini biliyor olmaları gerekmektedir.

Asimetrik Şifreleme ve Algoritmaları

Asimetrik şifreleme algoritması (ing. Asymmetric Encryption Algorithm), veri şifreleme ve çözme işleminde kullanılan anahtarın farklı olduğu algoritmalara verilen isimdir.

Şifreleme işleminde kullanılan anahtar, açık anahtar (ing. Public Key) olarak, şifre çözme işleminde kullanılan anahtar ise gizli anahtar (ing. Private Key) olarak tanımlanmaktadır. Bu iki anahtar birbirleri ile matematiksel olarak ilişkili değerlerden oluşmaktadır. Açık anahtar kullanılarak şifrelenen veri sadece gizli anahtar ile, gizli anahtar ile şifrelenen veri sadece açık anahtar ile çözülebilmektedir.

Şekil 42, Asimetrik Şifreleme İşlemi

Asimetrik algoritmaların genel kullanımı, açık ve gizli bölümlerden oluşan anahtar değerlerinden, açık anahtarın herkes tarafından, gizli anahtarın ise sadece veriyi çözecek kişi tarafından biliniyor olmasıdır.

Asimetrik şifreleme algoritmaları ile yapılan şifreleme işleminin, veri ve anahtar boyutuna bağlı olarak, simetrik algoritmalara göre çok daha yavaş olduğu unutulmamalıdır. Dolayısıyla, büyük miktarda verinin şifrelenmesinde asimetrik algoritmaların direkt kullanımı önerilmemektedir.

1978 yılında geliştirilen RSA, günümüzde de en çok kullanılan asimetrik şifreleme algoritmasıdır. Asimetrik algoritmalar pratikte SSL (Secure Socket Layer), PKI (Public Key Infrastructure) ve elektronik imza işlemlerinde kullanılmaktadır.

Algoritmaların Birlikte Kullanımı

Her iki algoritma tipinin sağladığı avantajlardan faydalanılarak, veri şifreleme işleminde simetrik ve asimetrik algoritmalar birlikte kullanılabilmektedir. Örneğin şifrelenmek istenen büyük miktarda veri simetrik algoritmalardan herhangi biri kullanılarak şifrelenmekte, sadece simetrik şifrele işleminde kullanılan anahtar değeri açık anahtar ile şifrelenmektedir. Böylece çok daha küçük anahtar bilgisinin şifrelenmesinde asimetrik algoritma kullanılmaktadır. Verinin simetrik, anahtarın asimetrik algoritma kullanılarak şifrelenmesi ardından, veriyi alan karşı taraf gizli anahtarı ile simetrik anahtara ve düz metne ulaşabilmektedir.

Şekil 43, Simetrik ve Asimetrik Algoritmaların Birlikte Kullanımı

Aktarılan yöntem pratikte SSL tarafından kullanılmaktadır. https ile başlayan bir web sitesinin ziyaretinde, yukarıda anlatılan işlemin yapıldığı SSL el sıkışması (ing. SSL Handshake) ile, internet tarayıcısı ve karşı sunucu arasında güvenli veri iletişimi sağlanabilmektedir.

Şifreleme Anahtarları

Gerek simetrik, gerek asimetrik şifreleme algoritmaları kullanımında en kritik konulardan birisi şifreleme anahtarlarının güvenliğidir. Şifreleme anahtarlarının kötü niyetli kişi tarafından ele geçirilmesi, kullanılan algoritma ne kadar güçlü olursa olsun, tüm veri güvenliğinin ele geçirilmesi anlamını taşımaktadır.

Günümüzde şifreleme anahtarları ve veri güvenliğinin üst düzeyde sağlanabilmesi amacıyla akıllı kart (ing. Smart Card) ve USB token cihazları kullanılmaktadır.

Özet Bilgi Kavramı ve Veri Bütünlüğü

Özet (ing. Hash) veri, büyük boyutlu veriden farklı özet algoritmaları (ing. Hash Algorithms) kullanılarak elde edilen daha küçük boyutlu olan veriye verilen isimdir.

Şekil 44, Özet Bilgi

Özet bilgi kullanımı ile hedeflenen veri bütünlüğünün sağlanabilmesidir. Özet bilginin karşı tarafa güvenle iletiminin sağlanması ile veriyi alan taraf, ulaşan verinin özet bilgisini hesaplamakta, kaynak tarafından iletilen özet bilgi değeri ile kendi hesapladığı değeri karşılaştırmaktadır. Özet bilgi değerlerinin aynı olmaması durumunda verinin değiştirildiği veya bozulduğu sonucu çıkmaktadır.

Özet bilgi algoritmaları, bir çeşit şifreleme algoritması gibi değerlendirilebilir. Özet bilgi algoritmalarının veriye uygulanması sonucu, boyutu orijinal veriden çok daha küçük olan farklı veri üretilmektedir. Buna rağmen, simetrik ve asimetrik şifreleme algoritmalarında şifreli metin değerinden düz metne ulaşılabilmesine rağmen, özet bilgiden gerçek veriye ulaşmak mümkün değildir.

Aşağıda günümüzde kullanılan özet bilgi algoritmaları, algoritmanın ürettiği özet bilginin boyutu ve açıklamaları gösterilmiştir.

| Üye Adı | Özet Bilgi Boyutu (Bit) | Açıklama |
| --- | --- | --- |
| MD5 | 128 | MD5, Message Digest algoritmasıdır. |
| SHA1 | 128 | SHA, Secure Hash Algorithm özet algoritmasıdır. |
| SHA256 | 256 | SHA, Secure Hash Algorithm özet algoritmasıdır. |
| SHA384 | 384 | SHA, Secure Hash Algorithm özet algoritmasıdır. |
| SHA512 | 512 | SHA, Secure Hash Algorithm özet algoritmasıdır. |

Tablo 118, Özet Algoritmaları ve Açıklamaları

Özet algoritmaları, veri boyutu ne olursa olsun yukarıdaki tabloda belirtilen uzunlukta özet bilgi üretmektedirler. Özet bilgi boyutunun yüksek olması özet algoritmasının daha güçlü olduğunu ifade etmektedir. Buna rağmen boyutun yüksek olmasının algoritma performansına olumsuz etkisi olabileceği unutulmalıdır.

Veri Güvenliği ve Özet Bilgi Kullanımı

Özet bilginin pratikte bir diğer kullanımı, veriyi saklamadan veri doğruluğunu kontrol edilebilmeyi sağlamaktır. Bazı durumlarda veri güvenliğini tam olarak sağlayabilmek için veriyi saklamamak en ideal çözüm olabilmektedir.

Örneğin uygulama yazılımlarında kimlik doğrulama (ing. Authentication) adımında amaç, kullanıcı tarafından sağlanan şifrenin doğruluğunu kontrol etmektir. Doğruluk kontrolü çoğunlukla, kullanıcı veritabanında saklanan kullanıcı adı / şifre ikili değerleri ile, kullanıcı tarafından sağlanan kullanıcı adı / şifre değerlerinin aynı olması esasına dayalıdır.

Yöntemin eşitlik esasına dayalı olması, kötü niyetli kişinin kullanıcı veritabanını ele geçirmesi ile sistemde bulunan bütün kullanıcıların kimlik doğrulama bilgilerine erişebilmesini beraberinde getirmektedir. Kullanıcı şifreleri yerine sadece şifrelerin özet bilgilerinin veritabanında saklanması, kötü niyetli kullanıcının verilere erişmesi durumunda dahi gerçek şifrele erişebilmesini engelleyecektir.

Şekil 45, Veri Güvenliği ve Özet Bilgi Kullanımı

Tuzlanmış Özet Bilgi

Tuzlanmış özet bilgi (ing. Salted Hash), orijinal veriye ekleme yapılması ile veri + eklenen veri sonucunda elde edilen özet değere verilen isimdir.

Yukarıdaki kimlik doğrulama örneğinde, gerçek şifreler veritabanında saklanmamasına rağmen, veritabanını ele geçiren kötü niyetli kişi saklanan şifre özet değerlerini kontrol ederek iki kullanıcının şifrelerinin aynı olduğunu bulabilir.

Tuzlanmış özet kullanımında, şifreye rasgele bir değer eklenerek aynı olan şifreler için farklı özet değerlerinin üretilmesi sağlanabilir. Eklenen değer, örneğin, veritabanı tablosunda ayrı bir sütun olarak tutulabilir.

Mesaj Doğrulama Kodu

Mesaj doğrulama kodu (ing. Message Authentication Code – MAC), iletilen mesajın bütünlüğü ve kimlik doğrulamasını yapmak amacıyla kullanılan bilgidir. Özet bazlı mesaj doğrulama kodları (ing. Hash based Message Authentication Code - HMAC), üretilen özet değerin şifrelenmesi ile özet değerin bütünlüğünü korumaktadırlar.

Mesaj doğrulama kodu desteği olan özet algoritmalar anahtarlı özet algoritmalar olarak (ing. Keyed Hash Algorithms) olarak tanımlanmaktadır. HMACSHA1 ve HMACTripleDES bu algoritmalara örnek olarak verilebilir.

Veri İmzalama

Veri güvenliğinin tam olarak sağlanmasını tamamlayan son konu inkar edilemezliktir. İnkar edilemezliği ve veri bütünlüğünü sağlayan veri imzalama (ing. Data Signature) kavramı, veri şifreleme ve özet bilgi kavramlarının birlikte kullanımıdır.

Şekil 46, Veri İmzalama

Veri imzalama, sadece veriyi imzalayan tarafta bulunan gizli anahtar ile veri özetinin herhangi bir asimetrik algoritma ile şifrelenmesini içermektedir. Böylece, özet bilgi kullanımı ile verinin bütünlüğü, gizli anahtar kullanımı ile şifreleme işlemini yapan kişinin bilinebilmesi sağlanmaktadır.

Veri imzalamanın veri şifrelemeden farklı olduğuna dikkat edilmelidir. İmzalama ile sadece özet bilgi şifrelenmektedir. İmzalama işleminin amacı, verinin şifrelenmesi değil bütünlüğünün korunması ve kaynağının bilinmesidir.

RSA ve DSA günümüzde en çok kullanılan veri imzalama algoritmalarıdır.

Dijital Sertifikalar ve Elektronik İmza

Elektronik imza, uluslar arası benimsenmiş kurallar, yöntemler, standartlar ve teknolojiler kullanılarak veri imzalamanın pratikte uygulanabilir şekilde kullanılmasıdır.

PKI olarak adlandırılan açık anahtar alt yapısı, gizli ve açık anahtarların dağıtım ve güvenliği, imzalama algoritmaları, güven ilişkileri gibi kavramların bütününün belirli standartlar ile yönetilebilmesine verilen isimdir.

Dijital Sertifikalar

Dijital sertifikalar (ing. Digital Certificate), PKI kapsamında farklı işlevlerin sağlanması amacıyla kullanılmaktadır. Örneğin dijital sertifika kişinin tanınmasını, dolayısıyla kimlik doğrulama işlevini, SSL güvenliğinin sağlanmasını veya veri şifreleme amacıyla kullanılabilir. Aynı sertifikanın birden fazla amaç için kullanımı da mümkündür.

Dijital sertifika, sertifika tarafından desteklenen işlevlerin yapılabilmesini sağlayan tüm bilgilerin bulunduğu yapıya verilen isimdir. Örneğin veri imzalamanın uygulanabilir şekilde sağlıklı olarak yapılabilmesi için gizli ve açık anahtarlar yeterli değildir. Anahtar değerlerine ek olarak anahtar sahibinin kim olduğunun, dolayısıyla imzalamayı yapan kişinin kim olduğunun bilinmesi gerekmektedir. Aynı zamanda imzalama işleminde kullanılan algoritma ismi de verinin doğrulanması için gerekmektedir.

Dijital sertifika içerisinde gizli anahtarın bulunma zorunluluğu yoktur. Örneğin veri imzalama işleminde gizli anahtar değeri gerekli iken, imzalanmış bir verinin doğrulanmasında sadece açık anahtarın bulunması yeterlidir.

Dijital sertifika aşağıda listelenen temel bilgileri içermektedir.

1. Sertifika konusu. Bu bilgi SSL sertifikaları için alan adı (ing. Domain Name) değeridir. Sertifika amacına göre kişi adı veya herhangi bir karakter dizi değer olabilir.
2. Sertifika seri numarası.
3. Kullanılan algoritmalar.
4. Sertifikanın amaçları. Örneğin sadece kimlik doğrulama veya veri imzalama, şifreleme.
5. Sertifika sahibinin açık anahtarı.
6. Sertifikanın geçerlilik süresi.
7. Sertifikayı veren kuruluşun imzası. İmza, tüm sertifika verisinin sertifikayı veren kuruluşun gizli anahtarı ile imzalanması ile oluşmakta ve sertifikanın geçerliliğini belirtmektedir.

Elektronik imzanın kanuni değer ifade ediyor olması, dijital sertifika dağıtımının belirli kurallarla yapılmasını zorunlu kılmaktadır. Dijital sertifika veren kuruluşlar sertifika otoriteleri (ing. Certificate Authority - CA) olarak adlandırılmaktadır. Dünyadaki popüler sertifika otoritelerine VeriSign ve GlobalSign örnek olarak verilebilir.

X509 sertifika formatı PKI kapsamında kullanılan dijital sertifika formatına verilen isimdir. Aşağıda Base64 ile kodlanmış örnek X509 sertifikası yapısı gösterilmiştir.

-----BEGIN CERTIFICATE-----

MIIDAjCCAmsCEH3Z/gfPqB63EHln+6eJNMYwDQYJKoZIhvcNAQEFBQAwgcExC

BgNVBAYTAlVTMRcwFQYDVQQKEw5WZXJpU2lnbiwgSW5jLjE8MDoGA1UECxMzQ

c3MgMyBQdWJsaWMgUHJpbWFyeSBDZXJ0aWZpY2F0aW9uIEF1dGhvcml0eSAtI

MTowOAYDVQQLEzEoYykgMTk5OCBWZXJpU2lnbiwgSW5jLiAtIEZvciBhdXRob

emVkIHVzZSBvbmx5MR8wHQYDVQQLExZWZXJpU2lnbiBUcnVzdCBOZXR3b3JrM

DTk4MDUxODAwMDAwMFoXDTI4MDgwMTIzNTk1OVowgcExCzAJBgNVBAYTAlVTM

-----END CERTIFICATE-----

Self Signed Certificate, sertifikayı oluşturan kişi tarafından aynı zamanda sertifikanın imzalanması ile üretilen sertifikaya verilen isimdir. Bu sertifikalar genellikle test amacıyla kullanılan sertifikalardır. .NET Framework SDK içerisinde dağıtılan makecert.exe ile bu sertifikaların üretilmesi mümkündür.

Microsoft tarafından PKI ile sertifika dağıtımı ve yönetimi amacıyla Microsoft Certificate Services yazılımı sunulmaktadır.

Güven İlişkisi

PKI ve dijital sertifika kullanımında en önemli konulardan birisi de taraflar arasındaki güven ilişkisidir (ing. Trust Relationship).

Güven ilişkisine örnek olarak nüfus cüzdanı verilebilir. Bir kişinin hangi ülke vatandaşı, adı soyadı gibi bilgiler nüfus cüzdanında bulunmaktadır. Geçerli bir nüfus cüzdanı kişinin gerçekte kim olduğunu anlamak için kullanılmaktadır. Buradaki güven ilişkisinin direkt kişiye değil, kişiye nüfus cüzdanını veren kuruluşa olduğuna dikkat edilmelidir.

Dijital sertifika nüfus cüzdanına benzetilebilir. Kişiyi dijital sertifika ile tanımlamada kullanılan güven ilişkisi sertifikayı veren kuruluşadır. Sertifika otoritesine güven duyulması, güven duyulan bir sertifika otoritesi tarafından sertifika kişiye verilmesi, kişiye güveni de beraberinde getirmektedir.

Güven ilişkisinin tek katman ile sınırlı olması zorunlu değildir. Kök sertifika otoritesi (ing. Root CA), güven ilişkisinde en üstte olan kuruluşu belirtmektedir. Bu kuruluş, bir başka kuruluşa sertifika dağıtma yetkisi verebilir. Bu durumda bir alt katmanda olan sertifika otoritesi alt sertifika otoritesi (Subordinate CA) olarak adlandırılmaktadır. Kök sertifika otoritesine olan güven, kök sertifika otoritesi tarafından yetki verilen bütün alt sertifika otoritelerine, dolayısıyla alt sertifika otoriteleri tarafından sertifika verilen bütün kişilere güveni beraberinde getirmektedir.

PKI geniş, detaylı ve gerçekleştirimi karmaşık bir mimaridir. PKI hakkında daha detaylı bilgi almak isteyen okuyucuların bu konuda yazılmış kitap ve kaynakları incelemeleri önerilmektedir.

Windows Kripto Mimarisi ve Kütüphaneler

Kriptografi fonksiyonlarının ve algoritmalarının gerçekleştirimi oldukça detaylı ve emek isteyen bir iştir. Kriptografi fonksiyonlarının bulunduğu ve Windows Kriptografi API (Win32 Cryptography Application Programming Interface) ve Açık SSL Kütüphanesi (OpenSSL Cryptography Library) Windows işletim sistemi için kullanılabilecek önemli kriptografi kütüphaneleridir.

Windows Kriptografi API

Microsoft tarafından kullanıma sunulan Windows kriptografi API veri şifreleme, özet bilgi, veri imzalama, şifreleme anahtarlarının saklanması ve yardımcı işlemlerin yapılabilmesini sağlayan fonksiyonlar içermektedir.

Kütüphane, kripto hizmet sağlayıcısı (ing. Cryptographic Service Provider – CSP) adı verilen yüklenebilir modüllerden oluşmaktadır. CSP kullanımı ile hedeflenen, uygulama yazılımları ile kripto işlemlerini sağlayan modüller arasına bir katman eklemektir.

Kripto hizmet sağlayıcıları Microsoft tarafından belirlenen standartlara uyularak farklı üreticiler tarafından da geliştirilebilmekte ve işletim sistemine yüklenebilmektedir. Böylece, kütüphaneyi kullanan uygulama yazılımlarında herhangi bir değişiklik gerekmeden farklı üreticiler tarafından sağlanan kripto fonksiyonlarına erişim mümkün olmaktadır.

Örneğin akıllı kart çözümleri sunan bir üretici şifreleme anahtarlarını akıllı kart üzerinde saklarken, bir başka üretici bu iş için işletim sisteminin yalıtılmış depolama alanını (ing. Isolated Storage) kullanabilir. Uygulama yazılımından Windows kriptografi API anahtar saklama fonksiyonu çağrıldığında, aktif CSP modülü otomatik olarak API tarafından kullanılmaktadır.

Sistemde yüklü CSP listesi sistem kayıt defteri içerisinde HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Cryptography\Defaults\Provider anahtarı altında bulunmaktadır.

Ekran Çıktısı 68, Sistemde Yüklü CSP Listesi

Uygulama yazılımı tarafından CSP adının belirtilmemesi durumunda kullanılacak CSP, ön tanımlı CSP (ing. Default CSP) olarak adlandırılmaktadır. Ön tanımlı CSP, Provider Types anahtarı altında tanımlanmıştır.

CSP Anahtar Veri Tabanı

Her CSP, şifreleme anahtarlarını saklamak için anahtar veritabanına (ing. Key Database) sahiptir. Anahtar veritabanı, anahtar kapsayıcıları (ing. Key Containers) ve her bir anahtar kapsayıcısı altında anahtar ikilileri şeklinde organize edilmiştir.

Anahtar kapsayıcıları kullanıcı veya bilgisayar bazında olabilmektedir. Anahtarlar kalıcı olarak veya oturum (ing. Session) bazında yönetilebilmektedir.

OpenSSL Kriptografi Kütüphanesi

OpenSSL kriptografi kütüphanesi açık kaynak kodlu olan ve farklı platform ve işletim sistemlerinde kullanılabilen kütüphanedir. OpenSSL hakkında daha fazla bilgiye www.openssl.org sitesinden ulaşılabilir.

OpenSSL kurulum dosyaları \OpenSSL klasöründe bulunmaktadır.

.NET Kripto Sınıfları ve Kullanımları

System.Security.Cryptography ad alanı içerisinde kripto işlemlerinin yapılabilmesini sağlayan sınıflar bulunmaktadır. Bulunan sınıfların çoğunluğu Windows kriptografi API fonksiyonlarını çağırmaktadır. Dolayısıyla bu sınıflar kullanılırken yönetilmeyen kod (ing. Unmanaged Code) çağrımının yapıldığı unutulmamalıdır.

Windows kriptografi API kullanan sınıflar CryptoServiceProvider soneki ile donatılmıştır. Örneğin RSACryptoServiceProvider, DESCryptoServiceProvider gibi.

Aktarılan bazı sınıflar System.Security.dll içerisinde bulunmaktadır. VS.NET ile çalışırken referans verilmesi gerekmektedir.

Bu bölümde, önceki konularda teorik düzeyde aktarılan işlemlerin .NET sınıfları ile yapılması hakkında bilgi verilecektir.

CSP Parametreleri

CspParameters sınıfı, kripto hizmet sağlayıcısına iletilmek istenen değerlerin belirtilebileceği sınıftır. .NET sınıf kütüphanesi içerisinde bulunan ve kripto hizmet sağlayıcı tarafından sunulan fonksiyonları çağıran sınıflar yapıcı metotlarında, isteğe bağlı olarak, bu sınıfı parametre olarak almaktadır.

Aşağıda CspParameters sınıfı üye bilgi ve açıklamaları gösterilmiştir.

| Üye Adı | Açıklama |
| --- | --- |
| KeyContainerName | Anahtar kapsayıcı adını belirtir. |
| ProviderName | Sağlayıcı adını belirtir. |
| Flags | Sağlayıcıya iletilen çeşitli parametreleri belirtir. |
| KeyPassword | Akıllı kart anahtarı ile ilişkilendirilmiş şifre değerini belirtir. |

Tablo 119, CspParameters Sınıfı Üye Bilgileri

Aşağıda CspParameters sınıfı örnek kullanımı gösterilmiştir.

C#

CspParameters p = **new** CspParameters();

p.ProviderName = "Microsoft Enhanced Cryptographic Provider v1.0";

RSACryptoServiceProvider rsa = **new** RSACryptoServiceProvider(p);

rsa.Encrypt(**new** **byte**[] {2, 3, 4}, **false**);

VB.NET

**Dim** p **As** CspParameters = **New** CspParameters()

p.ProviderName = "Microsoft Enhanced Cryptographic Provider v1.0"

**Dim** rsa **As** RSACryptoServiceProvider = **New** RSACryptoServiceProvider(p)

rsa.Encrypt(**New** **Byte**() {2, 3, 4}, **False**)

Kod 295, CspParameters Kullanım Örneği

Rasgele Sayı Üretimi

Veri güvenliğinin sağlanmasında rasgele sayı üretimi (ing. Random Number Generation) önem arz etmektedir. Rasgele sayı üretiminde en kritik konulardan birisi üretilen sayıların belirli alt ve üst limitler arasından düzgün dağılmış olmasıdır. Kritik bir diğer konu, bir sonraki rasgele sayının tahmin edilememesidir. Üretilecek sayının tahmin edilebilir olması, kötü niyetli kullanıcılar tarafından sisteme zarar vermek amacıyla kullanılabilmektedir.

Rasgele sayı üretiminde programcılar tarafından genellikle System.Random sınıfı kullanılmaktadır. Yüksek rasgelelik düzeyi isteyen kritik işlemlerde RNGCryptoServiceProvider sınıfının rasgele sayı üretiminde kullanımı önerilmektedir.

Aşağıdaki örnekte RNGCryptoServiceProvider sınıfı kullanılarak rasgele sayıların üretimi gösterilmiştir.

C#

RNGCryptoServiceProvider rng = **new** RNGCryptoServiceProvider();

**byte**[] randomBytes = **new** **byte**[1024];

// Rasgele üretilen byte değerleri ile doldur.

rng.GetBytes(randomBytes);

// Sadece sıfır olmayan değer ile doldur.

rng.GetNonZeroBytes(randomBytes);

VB.NET

**Dim** rng **As** RNGCryptoServiceProvider = **New** RNGCryptoServiceProvider()

**Dim** randomBytes(1023) **As** **Byte**

' Rasgele üretilen byte değerleri ile doldur.

rng.GetBytes(randomBytes)

' Sadece sıfır olmayan değer ile doldur.

rng.GetNonZeroBytes(randomBytes)

Kod 296, RNGCryptoServiceProvider Kullanım Örneği

Şifreleme Anahtarı Üretimi

Şifreleme anahtarları, yukarıda anlatılan yöntem kullanılarak rasgele şekilde üretilebilir. Üretilen anahtar daha sonra kullanılmak üzere güvenli bir ortamda saklanabilir.

Şifreleme anahtarı üretiminde kullanılabilecek alternatif bir yöntem de, anahtar boyutundan daha küçük bir veriden yola çıkılarak istenilen boyutta anahtar üretmektedir. Bir başka anlatımla, algoritmanın desteklediği en yüksek anahtar boyutundan daha küçük boyutta olan herhangi bir şifre değerinden anahtar değerini üretmektedir. Bu yöntem çoğunlukla kullanıcıdan alınan şifre bilgisi ile anahtarın oluşturulmasında kullanılmaktadır.

Bu amaçla, PBKDF1 algoritmasını kullanan PasswordDeriveBytes veya PBKDF2 algoritmasını kullanan Rfc2898DeriveBytes sınıfları bulunmaktadır.

Aşağıdaki örnekte Rfc2898DeriveBytes sınıfı kullanılarak on karakter olan şifre değerinden anahtar üretimi gösterilmiştir. Sınıf yapıcı metodu, anahtar üretiminde dağınıklığı daha üst düzeyde sağlamak amacıyla şifre bilgisine ek olarak tuz değerini parametre olarak almaktadır. Tuz değeri olarak en az 8byte uzunluğunda herhangi bir sabit değer vermek mümkündür.

C#

Console.WriteLine("Lütfen şifre giriniz:");

**string** sifre = Console.ReadLine();

**byte** [] salt = Encoding.ASCII.GetBytes(sifre);

Rfc2898DeriveBytes rfc = **new** Rfc2898DeriveBytes(sifre, salt);

**byte**[] anahtar = rfc.GetBytes(32);

Console.WriteLine("Üretilen anahtar:");

**foreach** (**byte** b **in** anahtar)

Console.Write("{0} ", b);

VB.NET

Console.WriteLine("Lütfen şifre giriniz:")

**Dim** sifre **As** **String** = Console.ReadLine()

**Dim** salt() **As** **Byte** = Encoding.ASCII.GetBytes(sifre)

**Dim** rfc **As** Rfc2898DeriveBytes = **New** Rfc2898DeriveBytes(sifre, salt)

**Dim** anahtar() **As** **Byte** = rfc.GetBytes(32)

Console.WriteLine("Üretilen anahtar:")

**For** Each b **As** **Byte** **In** anahtar

Console.Write("{0} ", b)

**Next**

Kod 297, Rfc2898DeriveBytes Sınıfı Kullanım Örneği

Kod çıktısı aşağıda gösterilmiş. Şifre değerinin aynı olması durumunda üretilen anahtar değerinin de her seferinde aynı olacağına dikkat edilmelidir.

Lütfen şifre giriniz: Sifrem1977

Üretilen anahtar:

51 71 223 68 178 165 107 31 30 246 53 114 25 144 254 49 225 53 239 232 146 70 73 129 170 125 235 253 167 17 213 85

Anahtar Saklama

Bilindiği gibi veri şifreleme işleminde en kritik konu şifreleme anahtarının nasıl saklandığıdır.

Güvenliğin üst düzeyde sağlanabilmesi için en ideal çözüm gerekmediği durumlarda anahtarın saklanmamasıdır. Örneğin anahtar, yukarıdaki örnekte olduğu gibi kullanıcıdan istenebilir. Bunun mümkün olmadığı durumlarda, anahtarın saklanması için Windows işletim sistemi tarafından sunulan veri koruma API (ing. Data Protection API-DPAPI) kullanımı önerilmektedir.

Şifreleme anahtarlarının kaynak kod içerisinde sabitler veya değişkenler kullanılarak saklanması kesinlikle önerilmemektedir.

Windows Veri Koruma API

Windows veri koruma API kullanılarak veri koruma, anahtar yönetiminin Windows işletim sistemi tarafından yapılması esasına dayalıdır. Bu yöntem özellikle şifreleme anahtarlarının korunması için ideal bir çözüm oluşturmaktadır.

DPAPI, LSA (Local Security Authority) tarafından sağlanan bir hizmettir. Şifreleme ve şifre çözme işlemleri Windows işletim sisteminin bir parçası olarak çalışan hizmet uygulaması tarafından yapılmaktadır ve şifreleme algoritması olarak 3DES kullanılmaktadır.

DPAPI, şifrelenmiş veri ve şifreleme anahtarının saklanmasını kullanıcı veya bilgisayar bazlı yapabilmektedir. Kullanıcı bazlı yapılan saklama, bir kullanıcının sisteme giriş yapması ile, ilgili kullanıcı hesabı altında çalışan tüm uygulamaların şifrelenmiş veriye erişebilmesini beraberinde getirmektedir. DPAPI tarafından şifrelemede kullanılan anahtar, işletim sistemine giriş yapan aktif kullanıcı şifresinden yukarıda anlatılan yöntem ile üretilmektedir. Bilgisayar bazlı saklama, bilgisayarda çalışan tüm uygulamaların şifrelenmiş veriye erişimini mümkün kılmaktadır.

DPAPI ile temelde hedeflenen, bir kullanıcı hesabı altında saklanan verinin, ancak kullanıcının sisteme giriş yapması ile kullanılabilir olabilmesidir. Buna rağmen entropi (ing. Entroph) adı verilen bilgi, zorunlu olmamakla birlikte, farklı uygulamaları ayırmak için kullanılabilmektedir.

Aşağıdaki örnekte verinin DPAPI ile saklanması ve saklanan verinin elde edilmesi gösterilmiştir. DPAPI tarafından simetrik bir algoritma olan 3DES kullanılmasına rağmen uygulama yazılımında veri şifrelemede herhangi bir anahtar iletilmediğine dikkat ediniz.

C#

**byte**[] deger = Encoding.UTF8.GetBytes("Test Değeri");

**byte** [] entrophy = Encoding.UTF8.GetBytes("Uygulamam");

**byte** [] sifrelenmisDeger = ProtectedData.Protect(deger, entrophy,   
 DataProtectionScope.CurrentUser);

**byte**[] duzDeger = ProtectedData.Unprotect(sifrelenmisDeger, entrophy,  
 DataProtectionScope.CurrentUser);

Console.WriteLine(Encoding.UTF8.GetString(duzDeger));

VB.NET

**Dim** deger() **As** **Byte** = Encoding.UTF8.GetBytes("Test Değeri")

**Dim** entrophy() **As** **Byte** = Encoding.UTF8.GetBytes("Uygulamam")

**Dim** sifrelenmisDeger() **As** **Byte** = ProtectedData.Protect(deger, entrophy, \_  
 DataProtectionScope.CurrentUser)

**Dim** duzDeger() **As** **Byte** = ProtectedData.Unprotect(sifrelenmisDeger, entrophy,\_  
 DataProtectionScope.CurrentUser)

Console.WriteLine(Encoding.UTF8.GetString(duzDeger))

Kod 298, DPAPI Kullanım Örneği

ProtectedMemory sınıfı, DPAPI kullanılarak verinin bilgisayar hafızasında şifrelenmesini ve çözülmesini sağlamak amacıyla kullanılmaktadır. ProtectedMemory sınıfı kullanımı ProtectedData sınıfı kullanımı ile benzerdir.

Simetrik Veri Şifreleme Sınıfları

Aşağıdaki listede simetrik algoritma sınıfları ve açıklamaları gösterilmiştir.

| Sınıf | Açıklama |
| --- | --- |
| RijndaelManaged | Tamamı yönetilen kod ile çalışan ve AES simetrik şifreleme algoritması gerçekleştirimini yapan sınıftır. Simetrik şifrelemede bu sınıfın kullanımı önerilmektedir. |
| DES | DES simetrik şifreleme algoritması gerçekleştirimini yapan sınıftır. |
| TripleDES | 3DES simetrik şifreleme algoritması gerçekleştirimini yapan sınıftır. |
| RC2 | Değişken anahtar boyutunu destekleyen ve DES standardında olan simetrik şifreleme algoritması gerçekleştirimini yapan sınıftır |

Tablo 120, Simetrik Veri Şifreleme Sınıf ve Açıklamaları

Simetrik veri şifreleme sınıfları soyut bir sınıf olan SymmetricAlgorithm sınıfını miras almaktadır.

SymmetricAlgorithm sınıfı üye bilgileri aşağıda gösterilmiştir.

| Üye Adı | Açıklama |
| --- | --- |
| BlockSize | Algoritma tarafından tek operasyonda işlenebilecek veri miktarını bit olarak ifade eder. |
| IV | İlk veri bloğunun korunması amacıyla kullanılan IV (Initialization Vector) değerini belirtir. |
| Key | Algoritma tarafından kullanılan simetrik anahtarı belirtir. Atanan simetrik anahtar değeri algoritmanın desteklediği boyutta olmalıdır. |
| KeySize | Simetrik anahtar boyutunu belirtir. |
| LegalKeySizes | Algoritma tarafından desteklenen anahtar boyutlarını döndüren özelliktir. |
| CreateDecryptor | Verinin çözülmesi için kullanılan ICryptoTransform arabirim gerçekleştirimini yapan nesneyi döndürür. |
| CreateEncryptor | Verinin şifrelenmesi için kullanılan ICryptoTransform arabirim gerçekleştirimini yapan nesneyi döndürür. |
| GenerateIV | Algoritma tarafından kullanılması istenen rasgele bir IV değeri üretir. |
| GenerateKey | Algoritma tarafından kullanılması istenen rasgele bir anahtar değeri üretir. |
| ValidKeySize | Parametre olarak iletilen anahtar boyutu değerinin algoritma tarafından desteklenip desteklenmediğini döndürür. |

Tablo 121, SymmetricAlgorithm Sınıfı Üye Bilgileri

IV, daha üst düzey güvenlik sağlamak amacıyla kullanılan değerdir.

Şifreleme Anahtarlarının Belirlenmesi

Şifreleme algoritma nesnesi oluşturulduğunda, algoritma tarafından desteklenen anahtar boyutunda otomatik olarak rasgele bir şifreleme anahtarı oluşturulmaktadır. Oluşturulan bu anahtar şifreleme işleminde kullanılabilir.

C#

RijndaelManaged aes = **new** RijndaelManaged();

Console.WriteLine("Anahtar boyutu (bit): {0}", aes.KeySize);

Console.WriteLine("Anahtar:");

**foreach** (**byte** b **in** aes.Key)

Console.Write("{0} ", b);

Console.WriteLine("Anahtar değeri boyutu: {0}", aes.Key.Length);

VB.NET

**Dim** aes **As** RijndaelManaged = **New** RijndaelManaged()

Console.WriteLine("Anahtar boyutu (bit): {0}", aes.KeySize)

Console.WriteLine("Anahtar:")

**For** **Each** b **As** **Byte** **In** aes.Key

Console.Write("{0} ", b)

**Next**

Console.WriteLine("Anahtar değeri boyutu: {0}", aes.Key.Length)

Kod 299, Simetrik Anahtarın Otomatik Olarak Oluşturulması

Kod çıktısı aşağıda gösterilmiştir.

Anahtar boyutu (bit): 256

Anahtar:

203 240 154 197 254 119 176 76 6 223 161 212 252 18 83 11 25 199 97 60 116 41 164 141 48 48 49 128 170 77 59 50

Anahtar değeri boyutu: 32

Otomatik üretilen anahtar değeri saklanarak, aynı anahtar daha sonra çözme işleminde de kullanılabilir.

Key özelliğine atanan değerin KeySize özelliği ile belirtilen boyutta olması gerekmektedir. Yukarıdaki örnekte anahtar boyutunun 256Bit (32Byte) olduğu görülmektedir.

Veri Şifreleme

ICryptoTransform arabirimi, düze metin – şifreli metin ve şifreli metin – düz metin dönüşümünün standart bir yapıya kavuşturulması için kullanılmaktadır. CreateEncryptor ve CreateDecryptor metotları bu arabirim gerçekleştirimini yapan nesneler döndürmektedir.

CryptoStream sınıfı, şifreleme ve şifre çözme işleminde kullanılan akım sınıfıdır. Aşağıdaki örnekte AES simetrik şifreleme algoritması kullanılarak veri şifreleme ve çözme işlemi gösterilmiştir.

C#

**private** **const** **string** IV = "0123456789ABCDRE";

**static** **void** Sifrele(**byte**[] anahtar) {

**using** (FileStream f = **new** FileStream(@"c:\x.dat", FileMode.Open)) {

RijndaelManaged aes = **new** RijndaelManaged();

aes.Key = anahtar;

aes.IV = Encoding.ASCII.GetBytes(IV);

ICryptoTransform t = aes.CreateEncryptor();

**byte**[] duzMetin = **new** **byte**[1024];

**using** (FileStream hedef = **new** FileStream(@"c:\x.sif", FileMode.Create)) {

**using** (CryptoStream cs = **new** CryptoStream(hedef, t,   
 CryptoStreamMode.Write)) {

**int** okunan;

**while** ((okunan = f.Read(duzMetin, 0, duzMetin.Length)) > 0) {

cs.Write(duzMetin, 0, okunan);

}

cs.Close();

}

hedef.Close();

}

f.Close();

}

}

**static** **void** Coz(**byte**[] anahtar)

{

**using** (FileStream f = **new** FileStream(@"c:\x.sif", FileMode.Open)) {

RijndaelManaged aes = **new** RijndaelManaged();

aes.Key = anahtar;

aes.IV = Encoding.ASCII.GetBytes(IV);

ICryptoTransform t = aes.CreateDecryptor();

**byte**[] düzMetin = **new** **byte**[1024];

**using** (FileStream hedef = **new** FileStream(@"c:\x.dat", FileMode.Create)) {

**using** (CryptoStream cs = **new** CryptoStream(f, t,   
 CryptoStreamMode.Read)) {

**int** okunan;

**while** ((okunan = cs.Read(düzMetin, 0, düzMetin.Length)) > 0) {

hedef.Write(düzMetin, 0, okunan);

}

cs.Close();

}

hedef.Close();

}

f.Close();

}

}

**public** **static** **void** Main()

{

Console.WriteLine("Şifre giriniz:");

**string** sifre = Console.ReadLine();

// Anahtar oluşturulmakta.

**byte**[] salt = Encoding.ASCII.GetBytes(sifre);

Rfc2898DeriveBytes rfc = **new** Rfc2898DeriveBytes(sifre, salt);

**byte** [] anahtar = rfc.GetBytes(32);

Console.WriteLine("Şifreleniyor ...");

Sifrele(anahtar);

Console.WriteLine("Çözülüyor ...");

Coz(anahtar);

}

VB.NET

**Private** **Const** IV **As** **String** = "0123456789ABCDRE"

**Shared** **Sub** Sifrele(**ByVal** anahtar() **As** **Byte**)

**Using** f **As** FileStream = **New** FileStream("c:\x.dat", FileMode.Open)

**Dim** aes **As** RijndaelManaged = **New** RijndaelManaged()

aes.Key = anahtar

aes.IV = Encoding.ASCII.GetBytes(IV)

**Dim** t **As** ICryptoTransform = aes.CreateEncryptor()

**Dim** duzMetin(1023) **As** **Byte**

**Using** hedef **As** FileStream = **New** FileStream("c:\x.sif", FileMode.Create)

**Using** cs **As** CryptoStream = **New** CryptoStream(hedef, t, \_  
 CryptoStreamMode.Write)

**Dim** okunan **As** **Integer** = f.Read(duzMetin, 0, duzMetin.Length)

**While** okunan > 0

cs.Write(duzMetin, 0, okunan)

okunan = f.Read(duzMetin, 0, duzMetin.Length)

**End** **While**

cs.Close()

**End** **Using**

hedef.Close()

**End** **Using**

f.Close()

**End** **Using**

**End** **Sub**

**Shared** **Sub** Coz(**ByVal** anahtar() **As** **Byte**)

**Using** f **As** FileStream = **New** FileStream("c:\x.sif", FileMode.Open)

**Dim** aes **As** RijndaelManaged = **New** RijndaelManaged()

aes.Key = anahtar

aes.IV = Encoding.ASCII.GetBytes(IV)

**Dim** t **As** ICryptoTransform = aes.CreateDecryptor()

**Dim** düzMetin(1023) **As** **Byte**

**Using** hedef **As** FileStream = **New** FileStream("c:\x.dat", FileMode.Create)

**Using** cs **As** CryptoStream = **New** CryptoStream(f, t, \_  
 CryptoStreamMode.Read)

**Dim** okunan **As** **Integer** = cs.Read(düzMetin, 0, düzMetin.Length)

**While** okunan > 0

hedef.Write(düzMetin, 0, okunan)

okunan = cs.Read(düzMetin, 0, düzMetin.Length)

**End** **While**

cs.Close()

**End** **Using**

hedef.Close()

**End** **Using**

f.Close()

**End** **Using**

**End** **Sub**

**Shared** **Sub** Main()

Console.WriteLine("Şifre giriniz:")

**Dim** sifre **As** **String** = Console.ReadLine()

**Dim** salt() **As** **Byte** = Encoding.ASCII.GetBytes(sifre)

**Dim** rfc **As** Rfc2898DeriveBytes = **New** Rfc2898DeriveBytes(sifre, salt)

**Dim** anahtar() **As** **Byte** = rfc.GetBytes(32)

Console.WriteLine("Şifreleniyor ...")

Sifrele(anahtar)

Console.WriteLine("Çözülüyor ...")

Coz(anahtar)

**End** **Sub**

Kod 300, AES Simetrik Şifreleme Sınıfı Kullanım Örneği

Listelenen kaynak kodlar \Bölüm 19\Simetrik klasöründe bulunmaktadır.

Asimetrik Veri Şifreleme Sınıfları

Asimetrik veri şifreleme sınıfları soyut bir sınıf olan AsymmetricAlgorithm sınıfını miras almaktadır.

RSACryptoServiceProvider sınıfı temel asimetrik veri şifreleme sınıfıdır ve 384Bit – 16384Bit arasındaki anahtar uzunluklarını desteklemektedir. Ön tanımlı anahtar uzunluğu 1024Bit olarak ayarlanmıştır. Yapıcı metoda farklı anahtar uzunluk değerleri parametre olarak iletilebilmektedir.

Kullanılan anahtar uzunluğunun yüksek olması daha üst düzey güvenlikle birlikte işlem performansında düşmeyi beraberinde getirmektedir. Ön tanımlı değer olan 1024Bit yeterli güvenliği sağlamaktadır.

Asimetrik Anahtarlar Oluşturma

RSAParameters sınıfı açık ve gizli anahtar değerlerini belirtmek için kullanılan sınıftır.

| Üye Adı | Açıklama |
| --- | --- |
| D | Gizli anahtarı belirtmektedir. |
| Exponent | Açık anahtarın kısa bölümünü ifade eden e değeridir. |
| Modulus | Açık anahtarın uzun bölümünü ifade eden n değeridir. |

Tablo 122, RSAParameters Sınıfı Üye Bilgileri

Bilindiği gibi gizli ve açık anahtar değerleri arasında matematiksel ilişki bulunmaktadır. Bu yüzden, simetrik algoritmalarda olduğu gibi, RSAParameters nesnesi oluşturularak gizli ve açık anahtar değerlerine herhangi rasgele değerin atanması mümkün değildir.

RSACryptoServiceProvidernesnesi oluşturulduğunda otomatik olarak açık ve gizli anahtar değerlerini oluşturulmaktadır. Bu değerlerin RSACryptoServiceProvider.ExportParameters metodu ile elde edilmesi mümkündür. Parametre olarak iletilen mantıksal değerin true olması durumunda açık anahtar değerleri ile birlikte gizli anahtar değeri de elde edilebilmektedir. Benzer şekilde, RSACryptoServiceProvider.ImportParameters metodu istenilen gizli açık anahtarın algoritmaya iletilmesini sağlamaktadır.

Aşağıdaki örnekte, RSACryptoServiceProvider nesnesi tarafından oluşturulan gizli ve açık anahtar değerlerinin elde edilmesi gösterilmiştir.

C#

RSACryptoServiceProvider rsa = **new** RSACryptoServiceProvider();

RSAParameters rsaParams = rsa.ExportParameters(**true**);

Console.WriteLine("Gizli anahtar:");

**foreach** (**byte** b **in** rsaParams.D)

Console.Write("{0} ", b);

VB.NET

**Dim** rsa **As** RSACryptoServiceProvider = **New** RSACryptoServiceProvider()

**Dim** rsaParams **As** RSAParameters = rsa.ExportParameters(**True**)

Console.WriteLine("Gizli anahtar:")

**For** **Each** b **As** **Byte** **In** rsaParams.D

Console.Write("{0} ", b)

**Next**

Kod 301, RSACryptoServiceProvider ve Gizli Açık Anahtar Değerleri

Kod çıktısı aşağıda gösterilmiştir.

Gizli anahtar:

140 243 138 104 202 50 182 121 232 162 50 207 228 125 204 109 229 210 213 4 139 255 63 208 20 72 29 179 250 44 68 236 99 72 251 64 84 164 53 179 165 221 43 236 12 225 222 20 51 146 169 154 184 9 220 9 108 89 21 142 114 243 55 73 208 0 4 251 110 22 125 21 197 38 196 249 97 188 174 199 45 190 6 255 112 109 211 127 153 206 70 217 46 206 164 87 94 119 79 107 165 87 162 194 126 113 46 18 210 109 25 123 50 83 65 59 131 90 149 3 224 14 247 227 176 58 251 113

Anahtarların Saklanması

Oluşturulan anahtarların saklanması için iki yöntem bulunmaktadır.

Kullanılabilecek birinci yöntem RSACryptoServiceProvider.ToXml metodu ile XML yapısında anahtar değerlerini elde etmektir. Bu metoda true değerinin parametre olarak iletilmesi gizli anahtar değerinin de döndürülmesini sağlamaktadır.

Saklanan değerlerin RSACryptoServiceProvider.FromXmlString metodu kullanılarak RSACryptoServiceProvider nesnesine iletimi mümkündür.

Aşağıdaki örnekte bu işlem gösterilmiştir.

C#

RSACryptoServiceProvider rsa = **new** RSACryptoServiceProvider();

**string** s = rsa.ToXmlString(**true**);

Console.WriteLine(s);

rsa.FromXmlString(s);

VB.NET

**Dim** rsa **As** RSACryptoServiceProvider = **New** RSACryptoServiceProvider()

**Dim** s **As** **String** = rsa.ToXmlString(**True**)

Console.WriteLine(s)

rsa.FromXmlString(s)

Kod 302, RSACryptoServiceProvider.ToXmlString Metodu Kullanımı

Gösterimi sadeleştirmek amacıyla sadece örnek değerler çıktıda gösterilmiştir.

<RSAKeyValue>

<Modulus>0rW5 . . .</Modulus>

<Exponent>AQAB . . .</Exponent>

<P>6/7Dt7Q9 . . .</P>

<Q>EFSV49+iF9+Zw== . . .</Q>

<DP>dmxPe6lw== . . .</DP>

<DQ>+xz3HU1wHZkqDEfMXw== . . . </DQ>

<InverseQ>1Be1FwuQmwd+5dgBZ . . .</InverseQ>

<D>kDYUvvA7cRog16k . . .</D>

</RSAKeyValue>

Kullanımı kolay ve pratik olan bu yöntem, gizli anahtar güvenliği ile ilgili herhangi bir çözüm sunmamaktadır.

Anahtarların saklanmasında kullanılabilecek bir diğer yöntem CSP anahtar kapsayıcılarıdır. Aşağıdaki örnekte bu yöntem kullanılarak anahtar yönetiminin yapılması gösterilmiştir. CspParameters.KeyContainerName özelliğine atanan değer anahtar kapsayıcı adını belirtmektedir.

C#

CspParameters csp = **new** CspParameters();

csp.KeyContainerName = "OrnekKapsayici";

RSACryptoServiceProvider rsa = **new** RSACryptoServiceProvider(csp);

rsa.PersistKeyInCsp = **true**;

RSAParameters rsaParams = rsa.ExportParameters(**true**);

Console.WriteLine("Gizli anahtar:");

**foreach** (**byte** b **in** rsaParams.D)

Console.Write("{0} ", b);

VB.NET

**Dim** csp **As** CspParameters = **New** CspParameters()

csp.KeyContainerName = "OrnekKapsayici"

**Dim** rsa **As** RSACryptoServiceProvider = **New** RSACryptoServiceProvider(csp)

rsa.PersistKeyInCsp = **True**

**Dim** rsaParams **As** RSAParameters = rsa.ExportParameters(**True**)

Console.WriteLine("Gizli anahtar:")

**For** Each b **As** **Byte** **In** rsaParams.D

Console.Write("{0} ", b)

**Next**

Kod 303, CSP Anahtar Kapsayıcı Kullanımı

Örnek her çalıştırıldığında aynı çıktıyı üretecektir. OrnekKapsayici isimli anahtar kapsayıcısının sistemde bulunması durumunda kapsayıcı içerisindeki değerler yüklenmektedir. Sistemde böyle bir kapsayıcı bulunması durumunda otomatik olarak oluşturulmaktadır.

Sistemde bulunan kapsayıcının silinmesi aşağıda gösterilmiştir.

C#

CspParameters csp = **new** CspParameters()*;*

csp.KeyContainerName = "OrnekKapsayici"*;*

RSACryptoServiceProvider rsa = **new** RSACryptoServiceProvider(csp)*;*

rsa.PersistKeyInCsp = **false***;*

rsa.Clear()*;*

VB.NET

**Dim** csp **As** CspParameters = **New** CspParameters()

csp.KeyContainerName = "OrnekKapsayici"

**Dim** rsa **As** RSACryptoServiceProvider = **New** RSACryptoServiceProvider(csp)

rsa.PersistKeyInCsp = **False**

rsa.Clear()

Kod 304, CSP Anahtar Kapsayıcısının Silinmesi

Farklı kripto hizmet sağlayıcı isimleri CspParameters.ProviderName özelliğine iletilerek, CSP tarafından akıllı kart veya bir başka ortamda anahtarların saklanması sağlanabilmektedir.

Veri Şifreleme

RSACryptoServiceProvider sınıfı kullanılarak veri şifreleme oldukça kolaydır. Aşağıda bununla ilgili örnek kod parçası gösterilmiştir.

C#

**string** duzMetin = "Test Değeri";

**byte**[] bytes = Encoding.UTF8.GetBytes(duzMetin);

RSACryptoServiceProvider rsa = **new** RSACryptoServiceProvider();

**byte**[] sifreliMetin = rsa.Encrypt(bytes, **false**);

bytes = rsa.Decrypt(sifreliMetin, **false**);

Console.WriteLine(Encoding.UTF8.GetString(bytes));

VB.NET

**Dim** duzMetin **As** **String** = "Test Değeri"

**Dim** bytes() **As** **Byte** = Encoding.UTF8.GetBytes(duzMetin)

**Dim** rsa **As** RSACryptoServiceProvider = **New** RSACryptoServiceProvider()

**Dim** sifreliMetin() **As** **Byte** = rsa.Encrypt(bytes, **False**)

bytes = rsa.Decrypt(sifreliMetin, **False**)

Console.WriteLine(Encoding.UTF8.GetString(bytes))

Kod 305, RSACryptoServiceProvider Sınıfı ile Veri Şifreleme

Bilindiği gibi asimetrik veri şifreleme algoritmaları, simetrik algoritmalara göre çok daha yavaştır. Bu yüzden asimetrik şifreleme algoritmaları ile büyük miktarda verinin şifrelenmesi önerilmemektedir. Büyük miktarda veriyi, asimetrik algoritmaların sağladığı üst düzey güvenliği de elde ederek şifrelemek için, simetrik şifreleme algoritmaları kullanılmalıdır. Simetrik şifrelemede kullanılan anahtar değerin asimetrik şifreleme algoritması ile korunması en ideal çözümü sağlayacaktır.

Özet Bilgi Sınıfları

.NET sınıf kütüphanesi içerisinde bulunan özet algoritma sınıfları aşağıda listelenmiştir.

Özet algoritmaları System.Security.Cryptography.HashAlgorithm soyut sınıfını miras almaktadır.

| Sınıf | Açıklama |
| --- | --- |
| MD5CryptoServiceProvider | MD5, Message Digest algoritması gerçekleştirim sınıfıdır. |
| RIPEMD160Managed | MD5 yerine kullanımı önerilen sınıftır. |
| SHA1CryptoServiceProvider, SHA256Managed, SHA384Managed, SHA512Managed | SHA, Secure Hash Algorithm özet algoritması gerçekleştirimleridir. |

Tablo 123, Özet Algoritma Sınıfları ve Açıklamaları

Aşağıda, mesaj doğrulama kodu desteği olan anahtarlı özet algoritma sınıfları bulunmaktadır.

Anahtarlı özet algoritmalar System.Security.Cryptography.KeyedHashAlgorithm soyut sınıfını miras almaktadır.

| Sınıf | Açıklama |
| --- | --- |
| HMACSHA1, HMACSHA256, HMACSHA384, HMACSHA512 | HMACSHA algoritması gerçekleştirim sınıflarıdır. |
| MACTripleDES | MACTripleDES algoritması gerçekleştirim sınıfıdır. |

Tablo 124, Özet Algoritma Sınıfları ve Açıklamaları

Özet Bilgi Algoritmaları Kullanımı

HashAlgorithm.ComputeHash metodu özet bilginin hesaplanması için kullanılan metottur. Hesaplanan özet bilgi değeri HashAlgorithm.Hash özelliği ile elde edilmektedir.

Aşağıdaki örnekte özet bilginin SHA256Managed sınıfı ile hesaplanması gösterilmiştir.

C#

**string** metin = "Test Değeri";

SHA256Managed hash = **new** SHA256Managed();

**byte**[] ozetDeger = hash.ComputeHash(Encoding.UTF8.GetBytes(metin));

Console.WriteLine("Ozet değer boyutu: {0}", ozetDeger.Length);

Console.WriteLine("Ozet değer base64: {0}",   
 Convert.ToBase64String(ozetDeger));

VB.NET

**Dim** metin **As** **String** = "Test Değeri"

**Dim** hash **As** SHA256Managed = **New** SHA256Managed

**Dim** ozetDeger() **As** **Byte** = hash.ComputeHash(Encoding.UTF8.GetBytes(metin))

Console.WriteLine("Ozet değer boyutu: {0}", ozetDeger.Length)

Console.WriteLine("Ozet değer base64: {0}", \_  
 Convert.ToBase64String(ozetDeger))

Kod 306, Özet Bilgi Değerinin Hesaplanması

Kod çıktısı aşağıda gösterilmiştir.

Ozet değer boyutu: 32

Ozet değer base64: CPGX0YjC1YlI+CQthu+f3RImun0gZLg62lhRU2jGBMo=

ComputeHash metodunun yüklenmiş sürümleri kullanılarak akım nesneleri parametre olarak iletilebilir.

Kullanıcı şifrelerinin yönetiminde özet bilgi kullanımı aşağıdaki kod parçasında örneklenmiştir.

C#

// Sifrem1977 değerinin SHA256 ile hesaplanmış değeridir. Gerçek şifre  
// yerine bu değer veritabanında saklanabilir.

**private** **const** **string** beklenenOzet =   
 "6cfaWMVoNkbDPa2mH/10PhwFyy3PWd3Pv96ipteuYUQ=";

**static** **void** Main() {

Console.Write("Şifre giriniz:");

**string** kullaniciSifresi = Console.ReadLine();

SHA256Managed hash = **new** SHA256Managed();

**byte**[] ozetDeger;

ozet Deger = hash.ComputeHash(Encoding.UTF8.GetBytes(kullaniciSifresi));

**string** kullaniciOzetDeger = Convert.ToBase64String(ozetDeger);

**if** (kullaniciOzetDeger == beklenenOzet)

Console.WriteLine("Şifre doğru");

**else**

Console.WriteLine("Şifre hatalı");

}

VB.NET

' Sifrem1977 değerinin SHA256 ile hesaplanmış değeridir. Gerçek şifre  
' yerine bu değer veritabanında saklanabilir.

**Private** **Const** beklenenOzet **As** **String** = \_  
 "6cfaWMVoNkbDPa2mH/10PhwFyy3PWd3Pv96ipteuYUQ="

**Shared** **Sub** Main()

Console.Write("Şifre giriniz:")

**Dim** kullaniciSifresi **As** **String** = Console.ReadLine()

**Dim** hash **As** SHA256Managed = **New** SHA256Managed()

**Dim** ozetDeger() **As** **Byte**

ozetDeger = hash.ComputeHash(Encoding.UTF8.GetBytes(kullaniciSifresi))

**Dim** kullaniciOzetDeger **As** **String** = Convert.ToBase64String(ozetDeger)

**If** (kullaniciOzetDeger = beklenenOzet) **Then**

Console.WriteLine("Şifre **do**ğru")

**Else**

Console.WriteLine("Şifre hatalı")

**End** **If**

**End** **Sub**

Kod 307, Özet Bilgi Kullanılarak Kimlik Doğrulaması Yapılması

Anahtarlı Özet Bilgi Kullanımı

Aşağıda anahtarlı özet bilgi kullanım örneği gösterilmiştir.

C#

**string** duzMetin = "Test değer";

**string** sifre = "Sifrem1977";

Rfc2898DeriveBytes rfc = **new** Rfc2898DeriveBytes(sifre,   
 Encoding.ASCII.GetBytes(sifre));

**byte** [] anahtar = rfc.GetBytes(64);

HMACSHA256 mac = **new** HMACSHA256(anahtar);

**byte**[] macDeger = mac.ComputeHash(Encoding.UTF8.GetBytes(duzMetin));

Console.WriteLine("Mac Değer Boyutu: {0}", macDeger.Length);

Console.WriteLine("Ozet değer base64: {0}", Convert.ToBase64String(macDeger));

VB.NET

**Dim** duzMetin **As** **String** = "Test değer"

**Dim** sifre **As** **String** = "Sifrem1977"

**Dim** rfc **As** Rfc2898DeriveBytes

rfc = **New** Rfc2898DeriveBytes(sifre, Encoding.ASCII.GetBytes(sifre))

**Dim** anahtar() **As** **Byte** = rfc.GetBytes(64)

**Dim** mac **As** HMACSHA256 = **New** HMACSHA256(anahtar)

**Dim** macDeger() **As** **Byte** = mac.ComputeHash(Encoding.UTF8.GetBytes(duzMetin))

Console.WriteLine("Mac Değer Boyutu: {0}", macDeger.Length)

Console.WriteLine("Ozet değer base64: {0}", Convert.ToBase64String(macDeger))

Kod 308, Anahtarlı Özet Bilgi Kullanım Örneği

Kod çıktısı aşağıda gösterilmiştir.

Mac Değer Boyutu: 32

Ozet değer base64:

vXuPhtFLFx2DuGbyKkFlpaYexr8dngzBMt6OyVmkzus=

Bilindiği gibi özet bilginin temel amacı veri bütünlüğünün sağlanmasıdır. Anahtarlı özet kullanımı, verinin iki taraf arasında iletiminde gerek veri bütünlüğünü gerek gizliliği sağlamaktadır.

Anahtarsız özet kullanımında, kötü niyetli kişi iletilen veriyi ve özet değeri birlikte değiştirebilir. Bu durumda karşı tarafın değişikliğin farkına varması mümkün değildir. Anahtarlı özet bilgi kullanımında, her iki tarafından da anahtar değerini biliyor olmaları verinin doğru şekilde gönderildiğini de garanti altına almaktadır.

Veri İmzalama

RSACryptoServiceProvider ve DSACryptoServiceProvider sınıfları veri imzalama amacıyla kullanılan sınıflardır. Her iki sınıf aynı işlevi farklı algoritmalarla sağlamaktadırlar.

DSACryptoServiceProvider sınıfı Data Signature Standart imzalama algoritmasını kullanmaktadır ve sadece veri imzalama işlevini sağlamaktadır. RSACryptoServiceProvider gerek veri şifreleme gerek imzalama amacıyla kullanılabilmektedir.

İmzalama

Aşağıdaki kod parçasında RSACryptoServiceProvider sınıfı kullanılarak veri imzalama işlemi örneklenmiştir. İmzalama işleminde kullanılan anahtarların önceki konularda anlatılan yöntemlerle saklanması mümkündür.

C#

**static** **void** Imzala()

{

// Dosya içeriğini oku.

**byte**[] dosya = File.ReadAllBytes(@"c:\x.dat");

CspParameters cspParams = **new** CspParameters();

cspParams.KeyContainerName = "OrnekKapsayici";

RSACryptoServiceProvider rsa = **new** RSACryptoServiceProvider(cspParams);

// İmzalamada kullanılan gizli anahtarı sakla.

rsa.PersistKeyInCsp = **true**;

// RSA algoritması ile SHA özet algoritması kullanarak imza bilgisini  
 // oluştur.

**byte**[] imza = rsa.SignData(dosya, **new** SHA1CryptoServiceProvider());

// İmza bilgisi ile birlikte veriyi imzalayan kişinin açık anahtarını  
 // kaydet.

File.WriteAllBytes(@"c:\x.imza", imza);

File.WriteAllText(@"c:\acik.Anahtar", rsa.ToXmlString(**false**));

}

VB.NET

**Shared** **Sub** Imzala()

' Dosya içeriğini oku.

**Dim** dosya() **As** **Byte** = File.ReadAllBytes("c:\x.dat")

**Dim** cspParams **As** CspParameters = **New** CspParameters()

cspParams.KeyContainerName = "OrnekKapsayici"

**Dim** rsa **As** RSACryptoServiceProvider

rsa = **New** RSACryptoServiceProvider(cspParams)

' İmzalamada kullanılan gizli anahtarı sakla.

rsa.PersistKeyInCsp = **True**

' RSA algoritması ile SHA özet algoritması kullanarak imza bilgisini  
 ' oluştur.

**Dim** imza() **As** **Byte** = rsa.SignData(dosya, **New** SHA1CryptoServiceProvider())

' İmza bilgisi ile birlikte veriyi imzalayan kişinin açık anahtarını  
 ' kaydet.

File.WriteAllBytes("c:\x.imza", imza)

File.WriteAllText("c:\acik.Anahtar", rsa.ToXmlString(**False**))

**End** **Sub**

Kod 309, RSA Veri İmzalama Örneği

Yukarıdaki kod parçasının çalıştırılması ile x.dat isimli dosyanın imza değeri x.imza isimli dosyaya yazılmaktadır. Aynı zamanda veriyi imzalayan tarafından açık anahtarı da acik.anahtar isimli dosyaya yazılmaktadır.

Üç dosyanın karşı tarafa iletilmesi ile, verinin (x.dat) kim tarafından üretildiği ve verinin yolda değişmediği (x.imza ve acik.anahtar dosyaları yardımıyla) anlaşılabilmektedir. Karşı tarafa veriyi imzalayan tarafın gizli anahtarının iletilmesinin gerekmediğine dikkat edilmelidir.

Aynı işlemin benzer şekilde DSACryptoServiceProvider sınıfıyla da yapılması mümkündür.

Doğrulama

Yukarıdaki örnek kod parçasında yapılan imzalama işleminin doğrulanmasının örneklendiği kod parçası aşağıda gösterilmiştir.

C#

**static** **void** Dogrula()

{

// İletilen dosyaları oku.

**byte**[] dosya = File.ReadAllBytes(@"c:\x.dat");

**byte** [] imza = File.ReadAllBytes(@"c:\x.imza");

**string** acikAnahtar = File.ReadAllText(@"c:\acik.Anahtar");

// Açık anahtar bilgisini algoritmaya ilet.

RSACryptoServiceProvider rsa = **new** RSACryptoServiceProvider();

rsa.FromXmlString(acikAnahtar);

// İmzanın geçerliliğini kontrol et.

**bool** sonuc = rsa.VerifyData(dosya, **new** SHA1CryptoServiceProvider(), imza);

**if** (!sonuc)

Console.WriteLine("Dosya imzası geçersizdir.");

**else**

Console.WriteLine("İmza geçerlidir");

}

VB.NET

**Private** **Shared** **Sub** Dogrula()

' İletilen dosyaları oku.

**Dim** dosya() **As** **Byte** = File.ReadAllBytes("c:\x.dat")

**Dim** imza() **As** **Byte** = File.ReadAllBytes("c:\x.imza")

**Dim** acikAnahtar **As** **String** = File.ReadAllText("c:\acik.Anahtar")

' Açık anahtar bilgisini algoritmaya ilet.

**Dim** rsa **As** RSACryptoServiceProvider = **New** RSACryptoServiceProvider

rsa.FromXmlString(acikAnahtar)

' İmzanın geçerliliğini kontrol et.

**Dim** sonuc **As** **Boolean**

Sonuc = rsa.VerifyData(dosya, **New** SHA1CryptoServiceProvider, imza)

**If** **Not** sonuc **Then**

Console.WriteLine("Dosya imzası geçersizdir.")

**Else**

Console.WriteLine("İmza geçerlidir")

**End** **If**

**End** **Sub**

Kod 310, İmzalanan Verinin Doğrulanması

Dijital Sertifika Kullanımı

Bilindiği gibi dijital sertifikalar, sertifika otoriteleri tarafından üretilmekte ve belirli kurallara göre dağıtılmaktadır. Her sertifika içerisinde, sertifikanın amacı, sertifika sahibini belirleyen konusu ve sertifikayı veren sertifika otoritenin imzası bulunmaktadır. Örneğin sertifika sadece veri imzalama için veya kişinin tanınması için kullanılabilmektedir.

Bu bölümde dijital sertifikaların pratikte kullanımı ve .NET sınıfları ile yönetimi konularında temel bilgiler verilecektir.

Sertifika Deposu

Sertifika kullanımında dikkat edilmesi gereken konulardan birisi de sertifikanın nerede ve nasıl depolandığıdır. Sertifikaların depolanması için en ideal çözümler akıllı anahtarlar veya USB token cihazlarıdır. Bu cihazlar sadece sertifikanın depolanmasını sağlamamakta, niteliklerine göre, gizli anahtarı cihaz dışına çıkartmadan şifreleme veya imzalama işlemini yapabilmektedirler.

Sertifikalar, Windows işletim sistemi tarafından disk üzerinde göreceli daha az güvenli olarak da saklanabilmektedir. Sertifika deposu (ing. Certificate Store) kullanıcı veya bilgisayar bazında sertifikaları saklayabilmektedir.

Bilgisayarda bulunan sertifikaların Microsoft yönetim konsolu (mmc.exe) yardımıyla görüntülenmesi ve yönetilmesi mümkündür. Bunun için Add / Remove Snap-in … komutu verilmeli ve Certificates uygulaması seçilmelidir.

Ekran Çıktısı 69, Certificates Uygulamasının Açılması

Aktif kullanıcıya ait sertifika deposu ekran çıktısı aşağıda gösterilmiştir.

Ekran Çıktısı 70, Aktif Kullanıcı Sertifika Deposu

Sertifika deposu, kişisel, güvenilen – güvenilmeyen sertifika otoriteleri gibi sertifikaların kategorize edilebilmelerini sağlayan mantıksal gruplardan oluşmaktadır.

Gizli anahtar içeren sertifikaların, genellikle, depolandığı bölüm Personal grubudur. Örneğin Trusted Root Certificate Authorities grubu, güvenilen kök sertifika otoritelerinin sertifikalarını depolamaktadır.

Sertifikaların Yüklenmesi ve Dışa Aktarımı

Sertifika otoritesi tarafından dağıtılan sertifika, sertifika otoritesinin bu konuda belirlediği kurallar çerçevesinde yüklenebilmektedir. Sertifika, en basit şekliyle, PKCS #12 (.pfx) yapısında olan bir dosyada gizli anahtarı ile birlikte dağıtılabilir.

Sertifikaların farklı dosya yapılarında dışa aktarılabilmeleri mümkündür. PKCS #12 yapısı gizli anahtar ile birlikte sertifikanın dışa aktarımını desteklemektedir. Gizli anahtarın dışa aktarımı güvenlik düzeyini önemli ölçüde düşürmektedir.

Sadece açık anahtar, DER veya Base64 kodlanmış x509 yapısında olan dosyalara aktarılabilir.

Örnek Sertifika Üretimi

.NET Framework SDK içerisinde dağıtılan makecert.exe test amaçlı sertifika üretmek için kullanılabilir. Aşağıdaki komutun çalıştırılması ile içerisinde gizli anahtarın da bulunduğu test sertifikası kişisel sertifika deposuna yüklenecektir.

makecert -pe -n   
"CN=Profesyonel Programlama Teknikleri.NET"   
-sky exchange -ss my

Komut çalıştırıldıktan sonra üretilen sertifika aşağıda gösterilmiştir.

Ekran Çıktısı 71, Örnek Test Sertifikası

Ekran çıktısında, sertifikanın üreticisine güvenilmediğini belirten mesaj görülmektedir. Bunun sebebi, sertifika üreticisinin açık anahtarını içeren sertifikanın Trusted Root Certificate Authorities grubunda bulunmamasıdır.

Sertifikanın dışa aktarımı için sertifika deposunda sertifika seçilip Export … komutu verilmelidir.

Ekran Çıktısı 72, Sertifikanın Dış Dosyaya Aktarımı

İşlem sonunda, gizli anahtar ile birlikte, sertifika istenilen dosyaya aktarılabilmektedir.

.NET Sınıfları ve Sertifika Yönetimi

System.Security.Cryptography.X509Certificates ad alanı içerisinde X509 sertifikalarının ve sertifika deposunun kullanım ve yönetimi amacıyla sunulmuş sınıflar bulunmaktadır.

X509Certificate2 sınıfı X509 sertifikasını simgeleyen sınıftır. Bu sınıf ile sertifika özelliklerine erişmek mümkündür.

X509Store sınıfı, sertifika deposundaki sertifikalara erişim ve sertifika deposunun yönetimi için kullanılmaktadır.

Sertifika Deposuna Erişim

Aşağıdaki örnek kod parçasında kişisel sertifika depolama grubu içerisinde bulunan sertifikaların temel özellikleri konsol ekranına yazdırılmaktadır.

C#

X509Store store = **new** X509Store(StoreName.My);

store.Open(OpenFlags.ReadOnly);

**foreach** (X509Certificate2 cer **in** store.Certificates)

{

Console.WriteLine("Sertifika Otoritesi: {0}", cer.Issuer);

Console.WriteLine("Sertifika Konusu: {0}", cer.Subject);

Console.WriteLine("Seri Numarası: {0}", cer.SerialNumber);

Console.WriteLine("Sertifika Özet Değeri: {0}", cer.Thumbprint);

**if** (cer.HasPrivateKey)

Console.WriteLine("Sertifika gizli anahtar içermekte.");

}

store.Close();

VB.NET

**Dim** store **As** X509Store = **New** X509Store(StoreName.My)

store.Open(OpenFlags.**ReadOnly**)

**For** **Each** cer **As** X509Certificate2 **In** store.Certificates

Console.WriteLine("Sertifika Otoritesi: {0}", cer.Issuer)

Console.WriteLine("Sertifika Konusu: {0}", cer.Subject)

Console.WriteLine("Seri Numarası: {0}", cer.SerialNumber)

Console.WriteLine("Sertifika Özet Değeri: {0}", cer.Thumbprint)

**If** cer.HasPrivateKey **Then**

Console.WriteLine("Sertifika gizli anahtar içermekte.")

**End** **If**

**Next**

Store.Close()

Kod 311, Sertifika Deposu Erişim Örneği

Bir önceki konuda yüklenen sertifikaya ait kod çıktısı aşağıda gösterilmiştir.

Sertifika Otoritesi: CN=Root Agency

Sertifika Konusu: CN=Profesyonel Programlama Teknikleri.NET

Seri Numarası: EBB1E5DC185A728B47EDA132C49DF882

Sertifika Özet Değeri: 3801C04059D97530F06840698D3079E4E790811B

Sertifika gizli anahtar içermekte.

Sertifika Nesnesi Kullanımı

Sertifika nesnesi, yukarıdaki örnekte olduğu gibi, sertifika deposu ile veya dışa aktarılmış sertifika dosyasından oluşturulabilir.

X509Certificate.CreateFromCertFile statik metodu dışa aktarılmış dosyadan sertifika nesnesi oluşturmak için kullanılmaktadır.

Sertifika nesnesinin pratikte en sık kullanılan özelliği AsymmetricAlgorithm nesnesi döndüren X509Certificate.PrivateKey özelliğidir. Aşağıdaki örnekte üretilen test sertifikası ile RSACryptoServiceProvider nesnesi oluşturulması gösterilmiştir. Nesne oluşturulduktan sonra kitapta anlatılan yöntemlerle veri şifrelemenin veya imzalamanın yapılması mümkündür.

C#

X509Store store = **new** X509Store(StoreName.My);

store.Open(OpenFlags.ReadOnly);

X509Certificate2Collection coll;

// Sertifika deposunda hash değeri ile sertifikayı bul.

coll = store.Certificates.Find(X509FindType.FindByThumbprint,  
 "3801C04059D97530F06840698D3079E4E790811B", **false**);

X509Certificate2 cert = coll[0];

// Asimetrik veri şifreleme nesnesini oıluştur.

RSACryptoServiceProvider rsa = (RSACryptoServiceProvider)cert.PrivateKey;

**string** duzMetin = "Test Değeri";

**byte**[] bytes = Encoding.UTF8.GetBytes(duzMetin);

// Sertifika içerisindeki gizli anahtar ile şifreleme yap.

**byte**[] sifreBytes = rsa.Encrypt(bytes, **false**);

VB.NET

**Dim** store **As** X509Store = **New** X509Store(StoreName.My)

store.Open(OpenFlags.**ReadOnly**)

**Dim** coll **As** X509Certificate2Collection

' Sertifika deposunda hash değeri ile sertifikayı bul.

coll = store.Certificates.Find(X509FindType.FindByThumbprint, \_  
 "3801C04059D97530F06840698D3079E4E790811B", **False**)

**Dim** cert **As** X509Certificate2 = coll(0)

' Asimetrik veri şifreleme nesnesini oıluştur.

**Dim** rsa **As** RSACryptoServiceProvider = **CType**(cert.PrivateKey, \_  
 RSACryptoServiceProvider)

**Dim** duzMetin **As** **String** = "Test Değeri"

**Dim** bytes() **As** **Byte** = Encoding.UTF8.GetBytes(duzMetin)

' Sertifika içerisindeki gizli anahtar ile şifreleme yap.

**Dim** sifreBytes() **As** **Byte** = rsa.Encrypt(bytes, **False**)

Kod 312, *Sertifika Nesnesi Kullanılarak Veri Şifreleme*

Kötümser yalnız tüneli görür, iyimser tünelin sonundaki ışığı görür, gerçekçi tünelle birlikte ışığı ve de gelecek treni görür.

J. Harris

Bölüm

20

Kimlik Doğrulama ve Yetki Yönetimi

K

ullanıcı yönetiminin bir bölümü olarak değerlendirilebilecek kimlik doğrulama ve yetkilendirme, kullanıcının sistemde tanınması ve yetkilerinin belirlenmesi ile yazılım güvenliğinin önemli bir parçasını oluşturmaktadır. Yazılım mimarisi belirlenirken veya geliştirme aşamasında bu konuda yapılabilecek hatalar kötü niyetli kişilerin kolaylıkla sisteme girebilmeleri ve sistem bütünlüğünü bozabilmeleri riskini beraberinde getirmektedir.

Bu bölümde, kimlik doğrulama ve yetkilendirme mekanizmaları ile .NET tarafından sunulan sınıflar hakkında bilgiler verilecektir.

Temel Kavramlar

Kimlik doğrulama (ing. Authentication), kullanıcının kim olduğunu ifade eden kavramdır. Yetkilendirme (ing. Authorization) ile, kullanıcının neler yapabileceği veya yapamayacağı belirlenmektedir.

Kimlik Doğrulama

Kimlik doğrulamanın temel amacı uygulama yazılımını kullanan kişinin kim olduğunu belirlemektir. Kimlik doğrulamanın farklı düzeylerde yapılabilmesi mümkündür. Örneğin bir web uygulamasında her kullanıcının ulaşabileceği sayfalar bulunurken bazı sayfalar sadece kimlik doğrulaması yapılmış kullanıcılara açık olabilir. Benzer şekilde uygulama, kimlik doğrulama işlemi başarıyla yapılmadan hiçbir kaynağa erişime izin vermeyebilir.

Anonim kullanıcı (ing. Anonymous User) kimlik doğrulaması yapılmamış kullanıcıya verilen isimdir.

Kimlik Doğrulama Yöntemleri

Uygulama yazılımlarında kimlik doğrulamanın üç temel yöntem ile yapılması mümkündür.

1. Kullanıcının ne bildiği
2. Kullanıcının neye sahip olduğu
3. Kullanıcının kim olduğu

Kullanıcının ne bildiği yöntemi, gizli değer veya değerlerin hem uygulama yazılımı hem kullanıcı tarafından bilinmesi esasına dayalıdır. Bu yöntem sağladığı kullanım kolaylı ile en çok tercih edilen kimlik doğrulama yöntemlerindendir. Örneğin kullanıcı adı şifresi ile kimlik doğrulamanın yapılması bu yöntemin en çok uygulanan modellerindendir. Burada yapılan işlem, gizli bilgiler olan kullanıcı adı ve şifresi değerlerinin uygulama yazılımı ve kullanıcı tarafından birlikte biliniyor olmasıdır.

Kullanıcının neye sahip olduğu, kullanıcını tarafından sunulan fiziksel bir materyale olan güven bazlıdır. Günlük hayatta kullanılan nüfus cüzdanları bu yöntemin örneklerindendir. Nüfus cüzdanının amacı kişinin kim olduğunu belirlemektir ve kişiyi tanımak isteyen otorite nüfus cüzdanını kişiyi tanımlamak amacıyla kullanmaktadır. Kişi tarafından taşınan akıllı veya tanıtıcı kartlar bu yönteme örnek olarak verilebilir.

Kullanıcının kim olduğu yöntemi biyolojik özellikler yardımıyla kullanıcıyı tanımlamak amacıyla kullanılmaktadır. Parmak izi veya retina bu yönteme örnek olarak verilebilir.

Üç temel yöntemin kimlik doğrulama amacıyla birlikte kullanımı da mümkündür. Örneğin kişi kendisini tanıtmak için akıllı kart ve şifre bilgisini birlikte iletebilir. Böylece ilk iki yöntemin birlikte başarıyla uygulanması ile kimlik doğrulama yapılabilir.

Yetkilendirme

Yetkilendirmenin temel amacı, kimlik doğrulaması yapılmış kullanıcının neleri yapabileceği ve neleri yapamayacağının belirlenmesidir. Örneğin bir web uygulamasında, kullanıcının yetki düzeyine göre farklı sayfalara, dolayısıyla uygulama içerisindeki işlevlere erişimi istenebilir.

En sık kullanılan yetki yöntemi rol bazlı yetkilendirmedir (ing. Role Based Authorization). Rol bazlı yetkilendirme, işlev yetkilerinin direkt sistem kullanıcıları yerine rollere atanmasını ve kullanıcının sahip olduğu rollere göre yetki kontrolünün yapılması esasına dayalıdır.

Yönetim kolaylığı ve diğer kazanımları ile, rollerin belirli bir hiyerarşik düzende organize edilmesi mümkündür. Hiyerarşik düzen, rollerinin birbirlerini kapsayabilmeleri ve roller arasındaki alt üst ilişkisine dayanmaktadır.

.NET Kimlik Doğrulama ve Yetki Mimarisi

System.Security.Principal ad alanı içerisinde bulunan IIdentity ve IPrincipal arabirimleri .NET kimlik doğrulama yetki mimarisinin temelini oluşturan arabirimlerdir.

Arabirim Bilgileri

IIdentity arabirimi kullanıcıyı ifade etmektedir.

C#

**public** **interface** IIdentity {

**string** AuthenticationType { **get**; }

**bool** IsAuthenticated { **get**; }

**string** Name { **get**; }

}

VB.NET

**Public** **Interface** IIdentity

**ReadOnly** **Property** AuthenticationType **As** **String**

**ReadOnly** **Property** IsAuthenticated **As** **Boolean**

**ReadOnly** **Property** Name **As** **String**

**End** **Interface**

Kod 313, System.Security.Principal.IIdentity Arabirim Tanımı

Aşağıda IIdentity arabirimi üye açıklamaları gösterilmiştir.

| Üye Adı | Açıklama |
| --- | --- |
| AuthenticationType | Kimlik doğrulama yöntemini belirten değerdir. |
| IsAuthenticated | Kullanıcının kimlik doğrulamasının yapılıp yapılmadığını belirten değerdir. |
| Name | Kullanıcı adını belirten değerdir. |

Tablo 125, System.Security.Principal.IIdentity Sınıfı Üye Bilgileri

IPrincipal kullanıcı yetkisini ifade etmek amacıyla kullanılan arabirimdir.

C#

**public** **interface** IPrincipal {

**bool** IsInRole(**string** role);

IIdentity Identity { get; }

}

VB.NET

**Public** **Interface** IPrincipal

**Function** IsInRole(**ByVal** role **As** **String**) **As** **Boolean**

**ReadOnly** **Property** Identity **As** IIdentity

**End** **Interface**

Kod 314, System.Security.Principal.IPrincipal Arabirim Tanımı

Aşağıda IPrincipal arabirimi üye açıklamaları gösterilmiştir.

| Üye Adı | Açıklama |
| --- | --- |
| Identity | Kimlik doğrulama arabirim gerçekleştirimini yapan nesneyi döndüren özelliktir. |
| IsInRole | Kullanıcının belirtilen role sahip olup olmadığını döndüren metottur. |

Tablo 126, System.Security.Principal.IIdentity Sınıfı Üye Bilgileri

IIdentity ve IPrincipal arabirimleri temel mimariyi oluşturmaktadır. Bu arabirim gerçekleştirimlerini yapan farklı sınıflar aracılığıyla farklı kimlik doğrulama ve yetki işlevleri sağlamak mümkündür.

İş Parçacıkları ve Yetkilendirme

Kimlik doğrulama ve yetki mimarisi iş parçacık bazında tasarlanmıştır. Bir başka anlatımla, her iş parçacığı kendisine ait kullanıcı bilgilerine sahiptir. Böylece çoklu iş parçacık mimarisine sahip uygulamalarda, farklı iş parçacıkları farklı yetkilerle çalışabilmektedir.

Thread.CurrentPrincipal statik özelliği, aktif iş parçacığı ile ilişkilendirilmiş yetki nesnesini döndürmektedir.

Ön Tanımlı Yetki Nesnesi

AppDomain.SetPrincipalPolicy ve AppDomain.SetThreadPolicy metotları, uygulama etki alanında iş parçacıkları ve yetki nesnesinin yönetim modelini belirtmektedir.

C#

**public** **enum** PrincipalPolicy {

UnauthenticatedPrincipal,

NoPrincipal,

WindowsPrincipal

}

**public** **void** SetPrincipalPolicy(PrincipalPolicy policy);

**public** **void** SetThreadPrincipal(IPrincipal principal);

VB.NET

**Public** **Enum** PrincipalPolicy

NoPrincipal

UnauthenticatedPrincipal

WindowsPrincipal

**End** **Enum**

**Public** **Sub** SetPrincipalPolicy(**ByVal** policy **As** PrincipalPolicy)

**Public** **Sub** SetThreadPrincipal(**ByVal** principal **As** IPrincipal)

Kod 315, İş Parçacıkları ve Uygulama Etki Alanı

Uygulama performansını artırmak amacıyla yeni oluşturulan iş parçacıkları için yetki nesneleri otomatik olarak oluşturulmamaktadır. SetPrincipalPolicy metodu, iş parçacıkları ile yetki nesnelerinin uygulama etki alanında nasıl ilişkilendirileceğini belirtmek amacıyla kullanılmaktadır.

PrincipalPolicy.NoPrincipal sayılabilir değeri iş parçacıklarının herhangi bir yetki nesnesi ile ilişkilendirilmemesi gerektiğini belirtmektedir. PrincipalPolicy.UnauthenticatedPrincipal sayılabilir değeri, iş parçacığı için kullanıcı doğrulaması yapılmamış anonim (IIdentity.IsAuthenticated özelliği false olan), PrincipalPolicy.WindowsPrincipal değeri ise Windows yetki nesnesinin (Windows Principal) oluşturulması gerektiğini ifade etmektedir.

Aşağıdaki kod parçasında aktif iç parçacığı hakkında bilgi alınması gösterilmiştir.

C#

**static** **void** Main() {

BilgiYaz();

}

**static** **void** BilgiYaz() {

IPrincipal p = Thread.CurrentPrincipal;

**if** (p != **null**) {

Console.WriteLine("Nesne Tipi: {0}", p.GetType().Name);

Console.WriteLine("Kullanıcı Adı: {0}", p.Identity.Name);

Console.WriteLine("Doğrulama Tipi: {0}", p.Identity.AuthenticationType);

} **else** Console.WriteLine("Yetki nesnesi bulunmamaktadır.");

}

VB.NET

**Shared** **Sub** Main()

BilgiYaz()

**End** **Sub**

**Shared** **Sub** BilgiYaz()

**Dim** p **As** IPrincipal = Thread.CurrentPrincipal

**Dim** o **As** **Object** = Thread.CurrentPrincipal

**If** **Not** p **Is** **Nothing** **Then**

Console.WriteLine("Nesne Tipi: {0}", o.GetType().Name)

Console.WriteLine("Kullanıcı Adı: {0}", p.Identity.Name)

Console.WriteLine("**Do**ğrulama Tipi: {0}", p.Identity. AuthenticationType)

**Else**

Console.WriteLine("Yetki nesnesi bulunmamaktadır.")

**End** **If**

**End** **Sub**

Kod 316, Aktif İş Parçacığı ve Yetki Nesnesi

Kod çıktısı aşağıda gösterilmiştir.

Nesne Tipi: GenericPrincipal

Kullanıcı Adı:

Doğrulama Tipi:

Aynı kod parçasının AppDomain.SetPrincipalPolicy metodu çağrılarak elde edilen çıktısı aşağıdaki gibi olmaktadır.

C#

**static** **void** Main() {

AppDomain.CurrentDomain.SetPrincipalPolicy(PrincipalPolicy.NoPrincipal);

BilgiYaz();

}

VB.NET

**Shared** **Sub** Main()

AppDomain.CurrentDomain.SetPrincipalPolicy(PrincipalPolicy.NoPrincipal)

BilgiYaz()

**End** **Sub**

Kod 317, AppDomain.SetPrincipalPolicy Kullanım Örneği

Yetki nesnesi bulunmamaktadır.

AppDomain.SetThreadPolicy metodu, oluşturulan yeni iş parçacıkları ile ilişkilendirilmek istenen yetki nesnesini belirtmek için kullanılmaktadır.

Hazır Yetki Sınıfları

.NET Framework içerisinde bulunan ve IIdentity ile IPrincipal arabirim gerçekleştirimlerini yapan yetki sınıfları aşağıda gösterilmiştir.

| Sınıf | Açıklama |
| --- | --- |
| GenericIdentity, GenericPrincipal | Miras alınarak veya direkt kullanım için hazırlanmış yetki sınıfıdır. |
| WindowsIdentity, WindowsPrincipal | Windows işletim sistemine giriş yapan kullanıcıyı ifade eden sınıftır. |
| FormsIdentity | ASP.NET Forms kimlik doğrulama sınıfıdır. |
| PassportIdentity | Microsoft Passport kimlik doğrulama sınıfıdır. |

Tablo 127, .NET Yetki Sınıfları ve Açıklamaları

Aşağıdaki örnek kod parçasında GenericIdentity ve GenericPrincipal nesneleri kullanılarak aktif iç parçacığına yetki nesnesinin atanması gösterilmiştir.

C#

GenericIdentity identity = **new** GenericIdentity("Kullanıcı");

**string** roller = **new** **string**[] { "Yetkili Kullanıcı", "Standart Kullanıcı" };

GenericPrincipal principal = **new** GenericPrincipal(identity, roller);

Thread.CurrentPrincipal = principal;

VB.NET

**Dim** identity **As** GenericIdentity = **New** GenericIdentity("Kullanıcı")

**Dim** roller **As** **String**() = {"Yetkili Kullanıcı", "Standart Kullanıcı"}

**Dim** principal **As** GenericPrincipal = **New** GenericPrincipal(identity, roller)

Thread.CurrentPrincipal = principal

Kod 318, GenericIdentity ve GenericPrincipal Kullanım Örneği

Programcı Tanımlı Yetki Sınıfları

IIdentity ve IPrincipal arabirim gerçekleştirimleri yapılarak veya GenericIdentity sınıfı miras alınarak kimlik doğrulama ve yetki sınıfları tanımlamak mümkündür.

Programcı tarafından oluşturulan yetki sınıflarına istenilen özellik ve metotlar eklenebilir. Örneğin kullanıcının e-posta adresi, sisteme son giriş IP adres değeri gibi her türlü bilginin sınıf içerisinde tanımlanması ve Thread.CurrentPrincipal özelliği ile elde edilmesi mümkündür.

Tasarım Bilgileri

Programcı tarafından yetki sınıflarının tanımlanması ve pratikte hatalı kullanımı uygulama yazılımlarında önemli güvenlik açıklarına yol açabilmektedir.

Kullanımda aşağıdaki adımların izlenmesi önerilmektedir.

1. Kimlik doğrulama ve yetki sınıfları bağımsız bir Assembly içerisinde miras alınamaz şekilde tanımlanmalıdır. Bunun için, kullanılan VS.NET çözümüne sınıf kütüphanesi tipinde yeni bir proje eklenebilir. Sınıfların miras alınamaz şekilde tanımlanması ile, kötü niyetli kullanıcının bu sınıfları miras alarak yeni sınıflar tanımlaması ve yetki denetimlerini geçmesi engellenmiş olacaktır.
2. Tanımlanan sınıfların yapıcı metotları sadece bulundukları Assembly içerisinden çağrılabilecek şekilde internal ve Friend (sırasıyla C# ve VB.NET) rezerve kelimeleri ile donatılmalıdır. Böylece sınıfların ve sınıflara ait nesnelerin istenilen Assembly içerisinden kullanılması sağlanacak ancak diğer Assembly dosyaları içerisinden yeni nesneler oluşturulamayacaktır.
3. KimlikYoneticisi isminde sınıf miras alınamaz şekilde aynı Assembly içerisinde tanımlanmalıdır. Sınıfa ait statik Principal özelliği aktif iş parçacığı ile ilişkilendirilmiş yetki nesnesini döndürmelidir. Dogrula isimli metot, parametre olarak aldığı kullanıcı adı ve şifresi ile yetki nesnesi oluşturmalıdır. Bu metoda iletilen kullanıcı adı ve şifresi doğrulanmalı, ancak doğrulama işlemi başarılı ise yetki nesnesi oluşturulmalıdır. Böylece dışarıdan yetki nesneleri oluşturmak için doğru kullanıcı adı ve şifre değerlerinin bu metoda iletimi zorunlu olmaktadır.
4. Assembly için imzalama ve güçlü isimlendirme kullanılmalıdır.
5. Kimlik doğrulama ve yetki sınıflarının Serialized öz niteliği ile donatılması önerilmemektedir. Böylece nesnelerin serilize edilebilmesi engellenecek ve kötü niyetli kullanıcının olası atakları engellenecektir.

Önerilen adımların izlenmesi güvenli yazılımlar geliştirmek için büyük önem taşımaktadır. Önerilen model şeması gösterilmiştir.

Şekil 47, Önerilen Kimlik Doğrulama ve Yetki Modeli

Yetki Sınıfları

Aşağıdaki kod parçasında örnek yetki sınıfları gösterilmiştir.

C#

**public** **sealed** **class** ProjeIdentity : IIdentity {

**private** **string** kullaniciAdi;

**private** **string** ePostaAdresi;

**public** **string** AuthenticationType {

**get** { **return** "Proje Kimlik Doğrulama Yöntemi"; } }

**public** **bool** IsAuthenticated { **get** { **return** **true**; } }

**public** **string** Name { **get** { **return** kullaniciAdi; } }

**public** **string** EPostaAdresi { **get** { **return** ePostaAdresi; } }

**internal** ProjeIdentity(**string** kullanici, **string** ePosta) {

**this**.kullaniciAdi = kullanici;

**this**.ePostaAdresi = ePosta;

}

}

**public** **sealed class** ProjePrincipal : IPrincipal {

**private** ProjeIdentity identity;

**private** List<**string**> roller;

**public** IIdentity Identity { **get** { **return** identity; } }

**public** **bool** IsInRole(**string** role) {

**return** roller.IndexOf(role) >= 0;

}

**internal** ProjePrincipal(ProjeIdentity identity, **string**[] roller) {

**this**.identity = identity;

**this**.roller = **new** List<**string**>(roller);

}

}

VB.NET

**Public NotInheritable** **Class** ProjeIdentity

**Implements** IIdentity

**Private** \_kullaniciAdi **As** **String**

**Private** \_ePostaAdresi **As** **String**

**Friend** **Sub** **New**(**ByVal** kullanici **As** **String**, **ByVal** ePosta **As** **String**)

**Me**.\_kullaniciAdi = kullanici

**Me**.\_ePostaAdresi = ePosta

**End** **Sub**

**Public** **ReadOnly** **Property** AuthenticationType() **As** **String**   
 **Implements** IIdentity.AuthenticationType

**Get**

**Return** "Proje Kimlik Dogrulama Yöntemi"

**End** **Get**

**End** **Property**

**Public** **ReadOnly** **Property** IsAuthenticated() **As** **Boolean**   
 **Implements** IIdentity.IsAuthenticated

**Get**

**Return** **True**

**End** **Get**

**End** **Property**

**Public** **ReadOnly** **Property** Name() **As** **String**   
 **Implements** IIdentity.Name

**Get**

**Return** \_kullaniciAdi

**End** **Get**

**End** **Property**

**Public** **ReadOnly** **Property** EPostaAdresi() **As** **String**

**Get**

**Return** \_ePostaAdresi

**End** **Get**

**End** **Property**

**End** **Class**

**Public** **NotInheritable Class** ProjePrincipal

**Implements** IPrincipal

**Private** \_identity **As** ProjeIdentity

**Private** \_roller **As** List(**Of** **String**)

**Friend** **Sub** **New**(**ByVal** identity **As** ProjeIdentity, **ByVal** roller() **As** **String**)

**Me**.\_identity = identity

**Me**.\_roller = **New** List(**Of** **String**)(roller)

**End** **Sub**

**Public** **ReadOnly** **Property** Identity() **As** IIdentity   
 **Implements** IPrincipal.Identity

**Get**

**Return** Identity

**End** **Get**

**End** **Property**

**Public** **Function** IsInRole(**ByVal** role **As** **String**) **As** **Boolean**   
 **Implements** IPrincipal.IsInRole

**Return** \_roller.IndexOf(role) >= 0

**End** **Function**

**End** **Class**

Kod 319, Örnek Kimlik Doğrulama ve Yetki Sınıfları

Kullanıcı Kimlik Doğrulama Sınıfı

Kullanımı önerilen kullanıcı kimlik doğrulama sınıf gerçekleştirimi aşağıda gösterilmiştir.

C#

**public** **static** **class** KimlikYoneticisi {

**public** **static** void Dogrula(**string** kullaniciAd, **string** sifre) {

// Bu metot içerisinde kullanıcı bilgilerinin ve rollerinin   
 // veritabanından okunması ve doğrulamasının yapılması mümkündür.

ProjeIdentity identity = **new** ProjeIdentity(kullaniciAd, "xxx@xxx.com")*;*

**string** [] roller = **new** **string** [] {"Yönetici"}*;*

IPrincipal principal = **new** ProjePrincipal(identity, roller)*;*

Thread.CurrentPrincipal = principal*;*

}

**public** **static** ProjePrincipal Principal {

**get** {

**return** (ProjePrincipal)Thread.CurrentPrincipal*;*

}

}

**public** **static** ProjeIdentity Identity {

**get** {

**return** (ProjeIdentity)Principal.Identity*;*

}

}

}

VB.NET

**Public** **NotInheritable** **Class** KimlikYoneticisi

**Public** **Shared** **Sub** Dogrula(**ByVal** kullaniciAd **As** **String**, \_  
 **ByVal** sifre **As** **String**)

' Bu metot içerisinde kullanıcı bilgilerinin ve rollerinin   
 ' veritabanından okunması ve doğrulamasının yapılması mümkündür.

**Dim** identity **As** ProjeIdentity

identity = **New** ProjeIdentity(kullaniciAd, "xxx@xxx.com")

**Dim** roller() **As** **String** = **New** **String**() {"Yönetici"}

**Dim** principal **As** IPrincipal = **New** ProjePrincipal(identity, roller)

Thread.CurrentPrincipal = principal

**End** **Sub**

**Public** **Shared** **ReadOnly** **Property** Principal() **As** ProjePrincipal

**Get**

**Return** **CType**(Thread.CurrentPrincipal, ProjePrincipal)

**End** **Get**

**End** **Property**

**Public** **Shared** **ReadOnly** **Property** Identity() **As** ProjeIdentity

**Get**

**Return** **CType**(Principal.Identity, ProjeIdentity)

**End** **Get**

**End** **Property**

**Private** **Sub** **New**()

**End** **Sub**

**End** **Class**

Kod 320, Kullanıcı Kimlik Doğrulama Sınıfı

Kullanım Bilgileri

Tanımlanan yetki ve kullanıcı kimlik doğrulama sınıfların örnek kullanımı aşağıda gösterilmiştir.

C#

**static** **void** Main() {

Console.WriteLine("Kullanıcı adı giriniz:");

**string** kullanici = Console.ReadLine();

Console.WriteLine("Şifre giriniz:");

**string** sifre = Console.ReadLine();

KimlikYoneticisi.Dogrula(kullanici, sifre);

BilgiYaz();

}

VB.NET

**Shared** **Sub** Main()

Console.WriteLine("Kullanıcı adı giriniz:")

**Dim** kullanici **As** **String** = Console.ReadLine

Console.WriteLine("Şifre giriniz:")

**Dim** sifre **As** **String** = Console.ReadLine

KimlikYoneticisi.Dogrula(kullanici, sifre)

BilgiYaz()

**End** **Sub**

Kod 321, Yetki ve Kullanıcı Kimlik Doğrulama Sınıfları Kullanımı

Kod çıktısı aşağıda gösterilmiştir.

Kullanıcı adı giriniz: XXX

Şifre giriniz: YYY

Nesne Tipi: ProjePrincipal

Kullanıcı Adı: XXX

Doğrulama Tipi: Proje Kimlik Doğrulama Yöntemi

Listelenen kaynak kodlar \Bölüm 20\Model klasöründe bulunmaktadır.

Kimlik Doğrulama ve İstisna Yönetimi

Kimlik doğrulamanın başarısız olması durumunda istisna fırlatılması ve kitapta daha önce anlatılan prensipler kullanılarak fırlatılan istisnanın gerektiği yerde yakalanması önerilmektedir.

System.Security.Authentication.AuthenticationException istisnası bu amaçla kullanılabilmektedir.

Yetki Kontrol Yöntemleri

Güvenli yazılımlar geliştirmenin önemli kriterlerinden olan kimlik doğrulama ve yetki mimarisinin önemli bir parçası da doğru yetki kontrol yöntemlerinin pratikte uygulanmasıdır.

Bilindiği gibi yansıma yardımıyla Assembly dosyaları içerisindeki sınıflara erişilebilmektedir. Kötü niyetli kişinin uygulama Assembly dosyalarını ele geçirmesi ile, dosyaların içerisindeki sınıf ve nesnelere erişimi mümkün olabilmektedir. İş sınıf üyeleri içerisinde yetki kontrolünün yapılmamış olması, kontrolün sadece uygulama girişinde, web sayfalarının veya Windows formlarının açılması aşamasında yapılıyor olması önemli güvenlik açıklarını da beraberinde getirmektedir.

Assembly dosyalarını korumak amacıyla Dotsfucator adı verilen yazılımlar kullanılabilmektedir. Bu yazılımlar, Assembly içerisinde bulunan sınıf, tip, nesne, yerel değişken isimlerini değiştirmekten sabit karakter dizi değerlerini şifrelemeye kadar bir dizi güvenlik önlemlerini sağlayabilmektedir.  
  
Assembly dosyalarını korumanın bir diğer yolu da .NET kod erişim güvenlik (Code Access Security – CAS) mimarisinin kullanılmasıdır. Geniş ve detaylı bir konu olan CAS ile, Assembly dosyaları arasındaki güven ilişkisi, kullanıcı yetkisine ek olarak kod yetkilerinin tanımlanabilmesi mümkündür. İzin arabirimi (IPermission) kullanılarak programcı tarafından da kod erişim mekanizması kullanılabilmektedir.  
  
Bu konuda hakkında bilgi verilen PrincipalPermission ve PrincipalPermissionAttribute sınıfları .NET kod erişim güvenlik mimarisinin bir parçasıdır.

Yetki kontrolünün gerekli tüm metotlar içerisinde yapılması önemle önerilmektedir. Metot bazında yetki kontrolünün yapılabilmesi amacıyla temel iki yöntem kullanılabilmektedir.

Özel Yetki Kontrolü

Özel yetki kontrolü, yetki kontrolünün yapıldığı sınıfların programcı tarafından geliştirilmesi ve gereken metotlar içerisinde bu sınıfın kullanılması modelidir.

Kullanıcının istenilen role sahip olmaması durumunda fırlatılacak istisna ile işlemin durması sağlanmaktadır. Aşağıda bu modelin uygulandığı örnek kod parçası gösterilmiştir.

C#

**public** **static** **class** KimlikYoneticisi {

. . .

**public** **static** **void** YetkiKontrolEt(**string** rol) {

IPrincipal principal = Thread.CurrentPrincipal;

// Kullanıcının belirtilen role sahip olmaması durumunda   
 // istisna fırlatılmaktadır.

**if** (!principal.IsInRole(rol))

**throw** **new** AuthenticationException("Yetersiz ayrıcalıklar");

}

}

**static** **void** KayitEkle() {

**try** {

KimlikYoneticisi.YetkiKontrolEt("Yönetici");

// Kullanıcının yetkisinin olmaması durumunda fırlatılan istisna  
 // işlemin durdurulmasını sağlayacaktır.

. . .

} **catch** (AuthenticationException) { . . . }

}

VB.NET

**Public** **NotInheritable** **Class** KimlikYoneticisi

. . .

**Public** **Shared** **Sub** YetkiKontrolEt(**ByVal** rol **As** **String**)

**Dim** principal **As** IPrincipal = Thread.CurrentPrincipal

' Kullanıcının belirtilen role sahip olmaması durumunda   
 ' istisna fırlatılmaktadır.

**If** **Not** principal.IsInRole(rol) **Then**

**Throw** **New** AuthenticationException("Yetersiz ayrıcalıklar")

**End** **If**

**End** **Sub**

**Shared** **Sub** KayitEkle()

**Try**

KimlikYoneticisi.YetkiKontrolEt("Yönetici")

' Kullanıcının yetkisinin olmaması durumunda fırlatılan istisna  
 ' işlemin durdurulmasını sağlayacaktır.

. . .

**Catch** exc **As** AuthenticationException

**End** **Try**

**End** **Sub**

Kod 322, Özel Yetki Kontrolü Örneği

.NET Kod Erişim Güvenlik Mimarisi

System.Security.Permissions ad alanı içerisinde bulunan PrincipalPermission ve PrincipalPermissionAttribute sınıfları rol bazlı yetki kontrolünün yapılabilmesini sağlamaktadır.

Öznitelik Kullanımı

PrincipalPermissionAttribute öz niteliği kullanılarak metot yetkilerinin belirlenmesi mümkündür. Declarative Role Based Security olarak tanımlanan bu yöntem, metotların öznitelikler ile donatılması ve CLR tarafından otomatik olarak yetki kontrolünün yapılmasını sağlamaktadır.

Aşağıdaki kod parçasında öznitelikler kullanılarak yetki kontrolü yapılması gösterilmiştir.

C#

// Metodun çağrımında sadece kullanıcı kimlik doğrulamasının yapılmış  
// olması yeterlidir. Kullanıcı kimlik doğrulaması yapılmamış bir  
// kullanıcı tarafından metot çağrıldığında CLR tarafından istisna  
// fırlatılacaktır. Yapılan işlem IIdentity.IsAuthenticated özelliğinin  
// değerinin CLR tarafından kontrolüdür.

[PrincipalPermission(SecurityAction.Demand, Authenticated=**true**)]

**static** **void** Test1()

{

}

// Sadece Yönetici rolüne sahip kullanıcı tarafından metodun   
// çağrılabilmesi sağlanmaktadır. Yapılan işlem IPrincipal.IsInRole metodunun // CLR tarafından çağrılması ve yetki kontrolünün yapılmasıdır.  
[PrincipalPermission(SecurityAction.Demand, Role = "Yönetici")]

**static** **void** Test2()

{

}

// Sadece kullanıcı adı Hasan olan kullanıcı tarafından metodun   
// çağrılabilmesi sağlanmaktadır. Yapılan işlem IIdentity.Name özellik  
// değerinin CLR tarafından kontrolüdür.

[PrincipalPermission(SecurityAction.Demand, Name="Hasan")]

**static** **void** Test3()

{

}

// Kullanıcının, Yönetici veya Yetkili Kullanıcılar rollerinden   
// herhangi birine sahip olması durumunda metodun çağrımına izin  
// verilmektedir.

[PrincipalPermission(SecurityAction.Demand, Role = "Yönetici")]

[PrincipalPermission(SecurityAction.Demand, Role = "Yetkili Kullanıcılar")]

**static** **void** Test4()

{

}

VB.NET

' Metodun çağrımında sadece kullanıcı kimlik doğrulamasının yapılmış  
' olması yeterlidir. Kullanıcı kimlik doğrulaması yapılmamış bir  
' kullanıcı tarafından metot çağrıldığında CLR tarafından istisna  
' fırlatılacaktır. Yapılan işlem IIdentity.IsAuthenticated özelliğinin  
' değerinin CLR tarafından kontrolüdür.

<PrincipalPermission(SecurityAction.Demand, Authenticated:=**True**)> \_

**Shared** **Sub** Test1()

**End** **Sub**

' Sadece Yönetici rolüne sahip kullanıcı tarafından metodun   
' çağrılabilmesi sağlanmaktadır. Yapılan işlem IPrincipal.IsInRole metodunun ' CLR tarafından çağrılması ve yetki kontrolünün yapılmasıdır.

<PrincipalPermission(SecurityAction.Demand, Role:="Yönetici")> \_

**Shared** **Sub** Test2()

**End** **Sub**

' Sadece kullanıcı adı Hasan olan kullanıcı tarafından metodun   
' çağrılabilmesi sağlanmaktadır. Yapılan işlem IIdentity.Name özellik  
' değerinin CLR tarafından kontrolüdür.

<PrincipalPermission(SecurityAction.Demand, Name:="Kullanıcı")> \_

**Shared** **Sub** Test3()

**End** **Sub**

' Kullanıcının, Yönetici veya Yetkili Kullanıcılar rollerinden   
' herhangi birine sahip olması durumunda metodun çağrımına izin  
' verilmektedir.

<PrincipalPermission(SecurityAction.Demand, Role:="Yönetici"), \_

PrincipalPermission(SecurityAction.Demand, Role:="Yetkili Kullanıcılar")> \_

**Shared** **Sub** Test4()

**End** **Sub**

Kod 323, Öznitelikler Kullanılarak Yetki Kontrolü

Yapılan yetki kontrolünün başarısız olması durumunda System.Security.SecurityException fırlatılmaktadır.

Öznitelik ile donatılmış bir metodun çağrımı ve fırlatılabilecek istisnanın yakalanması aşağıdaki kod parçasında gösterilmiştir.

| C# | VB.NET |
| --- | --- |
| **try** {  Test3();  }  **catch** (SecurityException){ . . . } | **Try**  Test3()  **Catch** exc **As** SecurityException  . . .  **End** **Try** |

Kod 324, Öznitelikler Kullanılarak Yetki Kontrolü ve İstisna Yönetimi

Yetki İzin Nesnesi Kullanımı

PrincipalPermission nesneleri ile, öznitelikler kullanılmadan, yetki kontrolünün yapılması mümkündür. Bu yöntem Imperative Role Based Security olarak tanımlanmaktadır.

Aşağıdaki kod parçasında PrincipalPermission nesnesi kullanımı gösterilmiştir.

C#

// Sadece kimlik doğrulama.

PrincipalPermission p1 = **new** PrincipalPermission(**null**, **null**, **true**);

p1.Demand();

// Yönetici rolü.

PrincipalPermission p2 = **new** PrincipalPermission(**null**, "Yönetici");

p2.Demand();

// Kullanıcı adının Hasan olması.

PrincipalPermission p3 = **new** PrincipalPermission(Hasan", **null**);

p3.Demand();

// Rolün Yönetici veya Yetkili Kullanıcı olması.

PrincipalPermission p4 = **new** PrincipalPermission(**null**, "Yetkili Kullanıcı");

IPermission p5 = p2.Union(p4);

p5.Demand();

VB.NET

' Sadece kimlik doğrulama.

**Dim** p1 **As** PrincipalPermission=**New** PrincipalPermission(**Nothing**, **Nothing**, **True**)

p1.Demand()

' Yönetici rolü.

**Dim** p2 **As** PrincipalPermission=**New** PrincipalPermission(**Nothing**, "Yönetici")

p2.Demand()

' Kullanıcı adının Hasan olması.

**Dim** p3 **As** PrincipalPermission=**New** PrincipalPermission("Hasan", **Nothing**)

p3.Demand()

' Rolün Yönetici veya Yetkili Kullanıcı olması.

**Dim** p4 **As** PrincipalPermission=**New** PrincipalPermission(**Nothing**, \_  
 "Yetkili Kullanıcı")

**Dim** p5 **As** IPermission = p2.Union(p4)

p5.Demand()

Kod 325, PrincipalPermission Sınıfı Kullanım Örneği

PrincipalPermission.Demand metodu yetki kontrolünü yapmaktadır. Kullanıcının istenilen role sahip olmaması durumunda System.Security.SecurityException fırlatılmaktadır.

PrincipalPermission.ToXml ve PrincipalPermission.FromXml metotları nesnenin Xml yapısında serilize ve de-serilize edilmesi amacıyla kullanılmaktadır. Aşağıdaki kod parçasında PrincipalPermission sınıfı önemli metotları kullanımı gösterilmiştir.

C#

**static** **void** Yaz(IPermission p) {

SecurityElement el = p.ToXml();

Console.WriteLine(el.ToString());

}

**static** **void** Main() {

PrincipalPermission p1 = **new** PrincipalPermission(**null**, "Yönetici");

PrincipalPermission p2 = **new** PrincipalPermission(**null**,   
 "Yetkili Kullanıcı");

Console.WriteLine("Xml yapısında p1:");

Yaz(p1);

Console.WriteLine("Xml yapısında p2:");

Yaz(p2);

IPermission p3 = p1.Union(p2);

Console.WriteLine("Xml yapısında p3 (p1 birleşim p2):");

Yaz(p3);

IPermission p4 = p3.Intersect(p1);

Console.WriteLine("Xml yapısında p4 (p3 kesişim p1):");

Yaz(p4);

}

VB.NET

**Shared** **Sub** Yaz(**ByVal** p **As** IPermission)

**Dim** el **As** SecurityElement = p.ToXml

Console.WriteLine(el.ToString)

**End** **Sub**

**Shared** **Sub** Main()

**Dim** p1 **As** PrincipalPermission = **New** PrincipalPermission(**Nothing**, \_   
 "Yönetici")

**Dim** p2 **As** PrincipalPermission = **New** PrincipalPermission(**Nothing**, \_  
 "Yetkili Kullanıcı")

Console.WriteLine("Xml yapısında p1:")

Yaz(p1)

Console.WriteLine("Xml yapısında p2:")

Yaz(p2)

**Dim** p3 **As** IPermission = p1.Union(p2)

Console.WriteLine("Xml yapısında p3 (p1 birleşim p2):")

Yaz(p3)

**Dim** p4 **As** IPermission = p3.Intersect(p1)

Console.WriteLine("Xml yapısında p4 (p3 kesişim p1):")

Yaz(p4)

**End** **Sub**

Kod 326, PrincipalPermission Sınıfı Metot Kullanım Örnekleri

Intersect, Union ve IsSubsetOf metotları kullanılarak kesişim, birleşim ve altküme kontrolünün yapılması mümkündür.

Aşağıda kod çıktısı gösterilmiştir.

Xml yapısında p1:

<IPermission class="System..." version="1">

<Identity Authenticated="true"

Role="Yönetici"/>

</IPermission>

Xml yapısında p2:

<IPermission class="System..." version="1">

<Identity Authenticated="true"

Role="Yetkili Kullanıcı"/>

</IPermission>

Xml yapısında p3 (p1 birleşim p2):

<IPermission class="System..." version="1">

<Identity Authenticated="true" Role="Yönetici"/>

<Identity Authenticated="true" Role="Yetkili Kullanıcı"/>

</IPermission>

Xml yapısında p4 (p3 kesişim p1):

<IPermission class="System..." version="1">

<Identity Authenticated="true" Role="Yönetici"/>

<Identity Authenticated="true"

Role=""/>

</IPermission>

İnsanlar, güller arasında dikenler bulunduğundan şikayet edeceklerine, dikenler arasında güller yaratıldığına şükretmelidir.

Mevlâna Celâleddin-i Rûmî

Bölüm

21

Konfigürasyon Yönetimi

K

onfigürasyon, uygulama yazılımlarının kullandıkları ayarların, tanımların ve yapılandırmanın dışarıdan parametrik olarak verilebilmesini ifade etmektedir.

Yazılım güvenliğinin de önemli bir parçası olan konfigürasyon yönetimi bu bölümün konusunu oluşturmaktadır.

Konfigürasyon ve Yazılım Güvenliği

Yazılım konfigürasyon bilgilerinin gizliliği yazılım güvenliğinin üst düzeyde sağlanabilmesi için büyük önem taşımaktadır. Kötü niyetli kişinin sadece konfigürasyon bilgilerini ele geçirerek uygulama hakkında detaylı bilgilere erişmesi ve bu bilgileri kullanarak sistemin geneli hakkında fikir sahibi olabilmesi mümkündür.

Örneğin yazılımın veritabanı bağlantısı yapmak için kullandığı parametreler içerisinde bulunan veritabanı yerleşimi, tipi, kullanıcı adı ve şifresi değerlerinin ele geçirilmesi ile kötü niyetli kişi veritabanına erişebilir, bu bilgilerden yola çıkarak uygulamanın kullandığı veritabanı sürümü, sunucu adı ve iç ağ yapısını tahmin edebilir. Konfigürasyon bilgilerinin kötü niyetli kişi tarafından değiştirilmesi ile uygulama istenildiği gibi çalışmayabilir veya beklenen sonucu üretmeyebilir.

Sisteme sızmaların ve olası kayıpların engellenmesi amacıyla, konfigürasyon yönetiminin her aşamasında konfigürasyon değerlerinin tutarlılığının kontrol edilmesi ve gizliliğinin sağlaması konusunda özen gösterilmelidir. Kitabın önceki konularında aktarılan veri şifreleme ve imzalama gibi yöntemlerin konfigürasyon bilgileri için kullanımı önerilmektedir.

.NET Konfigürasyon Mimarisi

Konfigürasyon yönetiminin standart bir yapıya kavuşturulabilmesini sağlamak amacıyla sunulan .NET konfigürasyon mimarisi hakkında bu konu başlığında bilgiler verilecektir.

Uygulama Konfigürasyon Dosyaları

Uygulama konfigürasyon dosyaları, konfigürasyon verisinin ve yapısının saklandığı Xml formatında olan dosyalardır. İki temel uygulama konfigürasyon dosya tipi bulunmaktadır.

Konsol ve Windows uygulamaları ön tanımlı adı app.config olan konfigürasyon dosyasını, ASP.NET web uygulamaları ve hizmetleri web.config isimli konfigürasyon dosyasını kullanmaktadır.

Windows uygulamaları VS.NET ile derlendiğinde uygulama konfigürasyon dosyası otomatik olarak uygulama çalışabilir dosyası (.exe) klasörüne kopyalanmaktadır. Örneğin uygulama.exe isimli bir dosyanın üretilmesi durumunda konfigürasyon dosyası uygulama.exe.config olarak değiştirilmekte ve çıktı klasörüne kopyalanmaktadır.

VS.NET ile Dosyalarının Eklenmesi

Projeye konfigürasyon dosyası eklemek için proje üzerinde sağ tıklanarak Add New Item … komutu verilmelidir.

Ekran Çıktısı 73, VS.NET ile Yeni Konfigürasyon Dosyası Eklenmesi

Konfigürasyon Dosya Yapısı

VS.NET ile oluşturulan konfigürasyon dosyası içeriği aşağıda gösterilmiştir.

<?xml version="1.0" encoding="utf-8" ?>

<configuration>

</configuration>

Kullanılan konfigürasyon bilgileri configuration elementleri arasında bulunmaktadır. Konfigürasyon dosyalarının amacı sadece uygulama tarafından kullanılan ayar ve tanımların saklanması değildir. Konfigürasyon dosyaları, kitabın .NET Assembly Bilgileri başlıklı bölümünde detaylarıyla aktarılan ve CLR tarafından kullanılan tanımların da saklandığı dosyalardır.

<?xml version="1.0" encoding="utf-8" ?>

<configuration>

<runtime>

<assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">

probing privatePath="Kütüphane"/>

</assemblyBinding>

</runtime>

</configuration>

Yukarıdaki örnekte Assembly bağlamanın nasıl yapılması istendiği bilgisi konfigürasyon dosyası yardımıyla CLR tarafından anlaşılmaktadır. Aşağıdaki örnekte ASP.NET mimarisinde web.config dosyası içerisinde system.web elementi içerisinde çeşitli tanımlar bulunmaktadır.

<?xml version="1.0"?>

<configuration>

<system.web>

<compilation debug="true" />

<authentication mode="Windows" />

</system.web>

</system.web>

</configuration>

Configuration elemeni içerisinde bulunan elementler konfigürasyon bölümleri (ing. Configuration Section) olarak tanımlanmaktadır. Örneklerde runtime ve system.web elementleri konfigürasyon bölümleridir. Konfigürasyon bölümleri içerisinde yeni konfigürasyon bölümleri tanımlanabilmektedir. Bu durumda üst konfigürasyon bölümü konfigürasyon bölüm grubu (ing. Configuration Section Group) olarak adlandırılmaktadır.

XML Elementleri ve Öznitelikleri Hakkında  
  
Amacı veriyi tanımlamak ve taşımak olan XML, element ve öznitelikler kullanarak bu işlevi sağlamaktadır. Aşağıdaki örnek XML kodunda elementler ve öznitelikler gösterilmiştir.  
  
<element1>  
 <element2 oznitelik="xxx"/>  
 <element3>  
 <altElement1>yyy< altElement1>  
 <element3>  
</ element1>  
  
Konfigürasyon bölümleri değerleri ön tanımlı olarak öznitelikler ile sağlanmaktadır. Bu değerler konfigürasyon özellikleri (ing. Configuration Property) olarak tanımlanmaktadır.

Temel Konfigürasyon Bölümleri

.NET tarafından iki temel konfigürasyon bölümü pratikte kullanım amacıyla tanımlanmıştır.

appSettings isimli konfigürasyon bölümü, uygulama ayar ve tanımlarının bulunabileceği bölümdür. Bu bölüm uygulama ayarları bölümü (ing. Application Settings Section) olarak tanımlanmaktadır.

connectionStrings isimli bölüm veritabanı bağlantı dizelerinin saklanabileceği bölümdür. Veritabanı bağlantı dizesi, uygulamanın veritabanına bağlanırken kullandığı parametrelerin bulunduğu karakter dizi değerdir. Bu değerin saklanması için konfigürasyon dosyası içerisinde özel bir bölümün bulunmasının temel sebeplerinden birisi, tasarım anında VS.NET ile yapılan veritabanı işlemleri için standart bağlantı dizesi saklama alanı oluşturulabilmesidir. Bu bölüm, veritabanı bağlantı dizeleri bölümü (ing. Database ConnectionStrings Section) olarak isimlendirilmektedir.

Aşağıdaki örnek konfigürasyon dosyasında her iki bölüm gösterilmiştir.

<?xml version="1.0"?>

<configuration>  
 <**appSettings**>

<add key ="Ayar1" value="Ayar1Deger"/>

<add key="Ayar2" value ="Ayar2Deger" />

</**appSettings**>

<**connectionStrings**>

<add name="SqlServerCs"

connectionString="Data Soure=xxx;User ..."/>

</**connectionStrings**>  
</configuration>

Konfigürasyon bilgilerinin korunması için configProtectedData konfigürasyon grubu kullanılmaktadır. Korunumlu konfigürasyon grubu olarak tanımlanan bu grup hakkında ilerleyen konularda bilgi verilecektir.

configSource özniteliği konfigürasyon bölümünün bir başka dosya içerisinde olabilmesini sağlamaktadır. Aşağıda örnek kullanım gösterilmiştir.

<configuration>

<appSettings **configSource**="ayarlar.config"/>

<connectionStrings **configSource**="dizeler.config"/>

</configuration>

Programcı Tanımlı Konfigürasyon Bölümleri

.NET konfigürasyon mimarisi programcı tarafından yeni konfigürasyon bölümlerinin ve gruplarının tanımlanabilmesini desteklemektedir. Programcı tanımlı konfigürasyon bölümlerinin nesnelere karşılık gelebilmesi ile konfigürasyon yönetimi nesne tabanlı olarak yapılabilmektedir.

configSections isimli konfigürasyon bölümü, konfigürasyon dosyası içerisinde bulunan programcı tanımlı bölümlerin hangi sınıflar ile yönetileceğini belirlemek ve böylece konfigürasyon yönetimi tarafından tanınmasını sağlamak amacıyla kullanılmaktadır.

Konfigürasyon yönetimi tarafından tanınmayan bir bölümün olması istisna fırlatılmasına sebep olacaktır.

Aşağıdaki örnekte programcı tanımlı konfigürasyon bölümü ve configSections kullanımı gösterilmiştir.

<configuration>

<**configSections**>

<**section** name="**AnaFormAyarlari**" type=". . ." />

<**sectionGroup** name ="**Uygulama.Ayarlari**" type =". . .">

<section name ="Genel" type =". . ."/>

</**sectionGroup**>

</**configSections**>

<**AnaFormAyarlari**

baslik="Form Başlığı"

maximize="true"

sonGiris="1999-01-01"

/>

<**Uygulama.Ayarlari**>

<**Genel** sonGirenKullanici="xxx"/>

</**Uygulama.Ayarlari**>

</configuration>

Programcı tanımlı konfigürasyon bölümünü ve grubunu belirtmek amacıyla configSections grubu içerisinde sırasıyla section ve sectionGroup elementleri bulunmaktadır. Name özniteliği bölüm veya grup adını, type ise bölümün / grubun yönetildiği sınıfı belirtmektedir.

Konfigürasyon Sınıfları

Konfigürasyon yönetiminde kullanılan temel sınıf System.Configuration ad alanı içerisinde statik olarak tanımlanmış olan ConfigurationManager ve Configuration sınıfıdır. Sınıflar System.Configuration.dll isimli Assembly içerisinde bulunmaktadır. VS.NET ile çalışırken proje referansları penceresi ile bu Assembly dosyasına referans verilmesi gerekmektedir.

.NET 2.0 ve üzeri sürümlerde ConfigurationManager sınıfının kullanımı önerilmektedir. Daha alt sürümleri (.NET 1.0 ve .NET 1.1) desteklemek amacıyla mscorlib.dll içerisinde System.Configuration ad alanı içerisinde de konfigürasyon yönetimi sınıfları bulunmaktadır. Yazılım geliştirirken bu sınıfların kullanılması derleme uyarısına yol açacaktır.

Konfigürasyon Yönetici Sınıfı

Aşağıdaki tabloda ConfigurationManager sınıfının önemli görülen üye bilgileri gösterilmiştir.

| Üye Adı | Açıklama |
| --- | --- |
| AppSettings | Uygulama konfigürasyon bilgileri bölümü değerlerini elde etmek için kullanılmalıdır. |
| ConnectionStrings | Veritabanı bağlantı dizesi konfigürasyon bölümü değerlerini elde etmek için kullanılmalıdır. |
| GetSection | Parametre olarak iletilen konfigürasyon bölümünü System.Object tipinde döndürür. |
| OpenExeConfiguration | Parametre olarak iletilen konfigürasyon dosyasını açar ve Configuration nesnesi olarak döndürür. |
| OpenMachineConfiguration | Sistem konfigürasyon dosyasını açar ve Configuration nesnesi olarak döndürür. |
| RefreshSection | Parametre olarak iletilen konfigürasyon bölümü bilgilerini günceller. |

Tablo 128, ConfigurationManager Sınıfı Üye Bilgileri

ASP.NET mimarisinde konfigürasyonun yönetimi için WebConfigurationManager isimli sınıf kullanılmaktadır. Üyeleri ve işlevleri benzer olan bu sınıf System.Web isimli Assembly dosyası içerisinde bulunmaktadır.

Aşağıda WebConfigurationManager sınıfının bazı üyeleri ve açıklamaları gösterilmiştir.

| Üye Adı | Açıklama |
| --- | --- |
| AppSettings | Uygulama konfigürasyon bilgileri bölümü değerlerini elde etmek için kullanılmalıdır. |
| ConnectionStrings | Veritabanı bağlantı dizesi konfigürasyon bölümü değerlerini elde etmek için kullanılmalıdır. |
| GetWebApplicationSection | Parametre olarak iletilen konfigürasyon bölümünü System.Object tipinde döndürür. |
| OpenWebConfiguration | Parametre olarak iletilen konfigürasyon dosyasını açar ve Configuration nesnesi olarak döndürür. |

Tablo 129, WebConfigurationManager Sınıfı Üye Bilgileri

Konfigürasyon Sınıfı

Konfigürasyon sınıfı uygulama konfigürasyon bilgilerini simgeleyen sınıftır. ConfigurationManager sınıfı statik metotları çağrılarak Configuration nesneleri oluşturulmaktadır.

Aşağıdaki tabloda Configuration sınıfının önemli görülen üye bilgileri gösterilmiştir.

| Üye Adı | Açıklama |
| --- | --- |
| AppSettings | Uygulama konfigürasyon bilgileri bölümü değerlerini elde etmek için kullanılmalıdır. |
| ConnectionStrings | Veritabanı bağlantı dizesi konfigürasyon bölümü değerlerini elde etmek için kullanılmalıdır. |
| EvaluationContext | Konfigürasyon nesnesi ile ilişkilendirilmiş bağlam bilgisini döndürür. |
| FilePath | Konfigürasyon dosya adını döndürür. |
| RootSectionGroup | Kök konfigürasyon bölüm grubunu ConfigurationSectionGroup nesnesi olarak döndürür. |
| Sections | Konfigürasyon bölümlerini ConfigurationSectionCollection nesnesi olarak döndürür. |
| SectionGroups | Konfigürasyon bölüm gruplarını ConfigurationSectionGroupCollection nesnesi olarak döndürür. |
| GetSection | Parametre olarak iletilen konfigürasyon bölümünü ConfigurationSection nesnesi olarak döndürür. |
| GetSectionGroup | Parametre olarak iletilen konfigürasyon bölüm grubunu ConfigurationSectionGroup nesnesi olarak döndürür. |
| Save | Konfigürasyon bilgilerini kaydeder. |
| SaveAs | Konfigürasyon bilgilerini farklı kaydeder. |

Tablo 130, System.Configuration.Configuration Sınıfı Üye Bilgileri

ConfigurationManager ve Configuration sınıflarının bazı üyeleri ortaktır. ConfigurationManager sınıfı ile hedeflenen uygulamanın aktif konfigürasyon verisine erişmektir. Configuration sınıfı istenilen konfigürasyon dosyasının yönetimi için kullanılmaktadır.

Yardımcı Sınıflar

Temel sınıf olan konfigürasyon yöneticisi ve konfigürasyon sınıflarına ek olarak aşağıda listelenen sınıflar konfigürasyon yönetimi amacıyla kullanılmaktadır.

| Sınıf | Açıklama |
| --- | --- |
| ConfigurationSection | Konfigürasyon bölümünü simgeleyen sınıftır. |
| ConfigurationSectionGroup | Konfigürasyon bölüm grubunu simgeleyen sınıftır. |
| ConfigurationProperty | Konfigürasyon özelliğini simgeleyen sınıftır. |
| ConfigurationElement | Konfigürasyon elementini simgeleyen sınıftır. |
| AppSettingsSection | Uygulama konfigürasyon bölümünü simgeleyen sınıftır. |
| ConnectionStringsSection | Veritabanı bağlantı dizeleri bölümünü simgeleyen sınıftır. |
| ProtectedConfigurationSection | Korunumlu konfigürasyon bölümünü simgeleyen sınıftır. |

Tablo 131, Yardımcı Configürasyon Sınıfları ve Açıklamaları

Konfigürasyon Yönetim İşlemleri

Aktarılan sınıflar ve mimari çerçevesinde pratikte uygulanabilecek konfigürasyon yönetim işlemleri ve detayları hakkında bu bölümde bilgiler sunulacaktır.

Konfigürasyon Bilgilerine Erişim

Konfigürasyon bilgilerine erişim amacıyla ConfigurationManager statik üyeleri kullanılabilir. Erişimde kullanılabilecek bir diğer yöntem de ConfigurationManager.OpenExeConfiguration veya WebConfigurationManager.OpenWebConfiguration metotları ile Configuration nesnesi oluşturulmasıdır.

Aşağıdaki kod parçasında her iki yöntemin de kullanımı örneklenmiştir.

C#

Configuration conf;

// Aşağıdaki satır ile aktif Configuration nesnesi elde edilmektedir.   
// Parametre olarak iletilebilecek dosya adı değeri ile istenilen   
// konfigürasyon dosyası açılabilir.

conf =ConfigurationManager.OpenExeConfiguration(ConfigurationUserLevel.None);

// Her iki yöntemle de erişim yapılabilir.

conf.AppSettings["Ayar1"];

ConfigurationManager.AppSettings["Ayar1"];

// Aşağıdaki satır ile ASP.NET web.config dosyası elde edilmektedir.  
// Parametre olarak iletilebilecek dosya adı değeri ile istenilen   
// konfigürasyon dosyası açılabilir.

conf = WebConfigurationManager.OpenWebConfiguration("~");

conf.AppSettings["Ayar1"];

WebConfigurationManager.AppSettings["Ayar1"];

VB.NET

**Dim** conf **As** Configuration

' Aşağıdaki satır ile aktif Configuration nesnesi elde edilmektedir.   
' Parametre olarak iletilebilecek dosya adı değeri ile istenilen   
' konfigürasyon dosyası açılabilir.

conf = ConfigurationManager.OpenExeConfiguration(ConfigurationUserLevel.None)

' Her iki yöntemle de erişim yapılabilir.

conf.AppSettings("Ayar1")

ConfigurationManager.AppSettings("Ayar1")

' Aşağıdaki satır ile ASP.NET web.config dosyası elde edilmektedir.  
' Parametre olarak iletilebilecek dosya adı değeri ile istenilen   
' konfigürasyon dosyası açılabilir.

conf = WebConfigurationManager.OpenWebConfiguration("~")

conf.AppSettings("Ayar1")

WebConfigurationManager.AppSettings("Ayar1")

Kod 327, Konfigürasyon Bilgileri Erişim Yöntemleri

Uygulama ve Veritabanı Bağlantı Dizeleri

Aşağıdaki konfigürasyon dosyası örneğinde iki adet uygulama, bir adet veritabanı bağlantı dize tanımı bulunmaktadır.

<configuration>

<**appSettings**>

<add key="Ayar1" value ="Ayar1Degeri"/>

<add key="Ayar2" value ="Ayar1Degeri2"/>

</**appSettings**>

<**connectionStrings**>

<add name="OracleCs"

connectionString="Data Source=..."

providerName="System.Data.OracleClient"/>

</**connectionStrings**>

</configuration>

Uygulama ve veritabanı konfigürasyon değerlerinin elde edilmesi aşağıdaki kod parçasında örneklenmiştir.

C#

**string** ayar1 = ConfigurationManager.AppSettings["Ayar1"];

**string** ayar2 = ConfigurationManager.AppSettings["Ayar2"];

ConnectionStringSettings baglanti;

baglanti = ConfigurationManager.ConnectionStrings["OracleCs"];

Console.WriteLine("Ayar1 değeri: {0}", ayar1);

Console.WriteLine("Ayar2 değeri: {0}", ayar2);

Console.WriteLine("Bağlantğ dizesi: {0}", baglanti.ConnectionString);

Console.WriteLine("Veri Sağlayıcısı: {0}", baglanti.ProviderName);

VB.NET

**Dim** ayar1 **As** **String** = ConfigurationManager.AppSettings("Ayar1")

**Dim** ayar2 **As** **String** = ConfigurationManager.AppSettings("Ayar2")

**Dim** baglanti **As** ConnectionStringSettings

baglanti = ConfigurationManager.ConnectionStrings("OracleCs")

Console.WriteLine("Ayar1 değeri: {0}", ayar1)

Console.WriteLine("Ayar2 değeri: {0}", ayar2)

Console.WriteLine("Bağlantı dizesi: {0}", baglanti.ConnectionString)

Console.WriteLine("Veri Sağlayıcısı: {0}", baglanti.ProviderName)

Kod 328, Uygulama ve Veritabanı Konfigürasyon Tanımlarına Erişim

Kod çıktısı aşağıda gösterilmiştir.

Ayar1 değeri: Ayar1Degeri

Ayar2 değeri: Ayar1Degeri2

Bağlantı dizesi: Data Source=...

Veri Sağlayıcısı: System.Data.OracleClient

Konfigürasyon Bölümleri

ConfigurationManager.GetSection veya Configuration.GetSection metodu istenilen konfigürasyon bölümünü elde etmek amacıyla kullanılmaktadır.

C#

**object** o = ConfigurationManager.GetSection("connectionStrings");

Configuration conf;

conf =configurationManager.OpenExeConfiguration(ConfigurationUserLevel.None);

ConfigurationSection sec = conf.GetSection("connectionStrings");

Console.WriteLine(o.GetType().Name);

Console.WriteLine(sec.GetType().Name);

Console.WriteLine("Bölüm bilgileri:");

Console.WriteLine("Adı : {0}", sec.SectionInformation.SectionName);

Console.WriteLine("Tipi: {0}", sec.ElementInformation.Type.Name);

Console.WriteLine("Satır Numarası: {0}", sec.ElementInformation.LineNumber);

Console.WriteLine("Korunumlu mu ?: {0}", sec.SectionInformation.IsProtected);

VB.NET

**Dim** o **As** **Object** = ConfigurationManager.GetSection("connectionStrings")

**Dim** conf **As** Configuration

conf = ConfigurationManager.OpenExeConfiguration(ConfigurationUserLevel.None)

**Dim** sec **As** ConfigurationSection = conf.GetSection("connectionStrings")

Console.WriteLine(o.GetType().Name)

Console.WriteLine(sec.GetType().Name)

Console.WriteLine("Bölüm bilgileri:")

Console.WriteLine("Adı : {0}", sec.SectionInformation.SectionName)

Console.WriteLine("Tipi: {0}", sec.ElementInformation.Type.Name)

Console.WriteLine("Satır Numarası: {0}", sec.ElementInformation.LineNumber)

Console.WriteLine("Korunumlu mu ?: {0}", sec.SectionInformation.IsProtected)

Kod 329, Konfigürasyon Bölümlerine Erişim

Kod parçasının örnek konfigürasyon dosyası için ürettiği çıktı aşağıda gösterilmiştir.

ConnectionStringsSection

ConnectionStringsSection

Bölüm bilgileri:

Adı : connectionStrings

Tipi: ConnectionStringsSection

Satır Numarası: 7

Korunumlu mu ?: False

Configuration.Sections özelliği konfigürasyon dosyası içerisindeki bölümleri listelemek amacıyla kullanılabilir.

Konfigürasyon Bölüm Grupları

Konfigürasyon bölüm gruplarına erişmek için Configuration.GetSectionGroup metodu kullanılmalıdır.

C#

Configuration conf;

conf =ConfigurationManager.OpenExeConfiguration(ConfigurationUserLevel.None);

ConfigurationSectionGroup grp = conf.GetSectionGroup("configuration");

Console.WriteLine("Bölüm Grubu bilgileri:");

Console.WriteLine("Adı : {0}", grp.SectionGroupName);

Console.WriteLine("Tipi: {0}", grp.Type);

Console.WriteLine("Grup bölümleri:");

**foreach** (ConfigurationSection sec **in** grp.Sections) {

Console.WriteLine("Bölüm adı: {0}", sec.SectionInformation.Name);

Console.WriteLine("Bölüm tipi: {0}", sec.SectionInformation.Type);

}

VB.NET

**Dim** conf **As** Configuration

conf = ConfigurationManager.OpenExeConfiguration(ConfigurationUserLevel.None)

**Dim** grp **As** ConfigurationSectionGroup = conf.GetSectionGroup("configuration")

Console.WriteLine("Bölüm Grubu bilgileri:")

Console.WriteLine("Adı : {0}", grp.SectionGroupName)

Console.WriteLine("Tipi: {0}", grp.Type)

Console.WriteLine("Grup bölümleri:")

**Dim** sec **As** ConfigurationSection

**For** **Each** sec **In** grp.Sections

Console.WriteLine("Bölüm adı: {0}", sec.SectionInformation.Name)

Console.WriteLine("Bölüm tipi: {0}", sec.SectionInformation.Type)

**Next**

Kod 330, Konfigürasyon Gruplarına Erişim

Configuration.GetSectionGroups özelliği konfigürasyon dosyası içerisindeki bölüm gruplarını listelemek amacıyla kullanılabilir.

Konfigürasyon Bilgilerini Kaydetme

Configuration.Save ve Configuration.SaveAs metotları Configuration nesnesi içerisinde tutulan konfigürasyon bilgilerinin kaydedilmesi amacıyla kullanılmaktadır.

Aşağıdaki kod parçasında yeni bir uygulama konfigürasyon tanımı eklenmesi ve farklı bir dosyaya konfigürasyonun kaydedilmesi gösterilmiştir.

C#

Configuration conf;

conf =ConfigurationManager.OpenExeConfiguration(ConfigurationUserLevel.None);

conf.AppSettings.Settings.Add("Ayar3", "Ayar3Degeri");

conf.SaveAs(@"c:\konfigurasyon.config");

VB.NET

**Dim** conf **As** Configuration

conf = ConfigurationManager.OpenExeConfiguration(ConfigurationUserLevel.None)

conf.AppSettings.Settings.Add("Ayar3", "Ayar3Degeri")

conf.SaveAs("c:\konfigurasyon.config")

Kod 331, Konfigürasyon Bilgilerini Kaydetme Kod Örneği

Programcı Tanımlı Konfigürasyon Bölümleri

.NET konfigürasyon mimarisinin en güçlü yönlerinden birisi de programcı tarafından yeni konfigürasyon bölümleri oluşturulabilmesi ve oluşturulan bölümün kod içerisinden kolaylıkla kullanılabilmesidir.

Yeni konfigürasyon bölümü oluşturmak için soyut bir sınıf olan ConfigurationSection sınıfının miras alınması yeterlidir.

Konfigürasyon özelliklerini iki modelde belirtmek mümkündür. İlk model sınıf içinde tanımlanan özelliklerin ConfigurationPropertyAttribute özniteliği ile donatılmasıdır. İkinci model, ConfigurationSection.ConfigurationPropertyCollection özelliğini örterek konfigürasyon özelliklerini ata sınıfa iletmektir.

ConfigurationProperty sınıfı üye bilgi ve açıklamaları aşağıda gösterilmiştir.

| Üye Adı | Açıklama |
| --- | --- |
| DefaultValue | Ön tanımlı değeri belirtmektedir. |
| IsRequired | Konfigürasyon özelliğinin zorunlu olup olmaması gerektiğini belirtmektedir. |
| Name | Konfigürasyon özellik adını belirtmektedir. |

Tablo 132, ConfigurationProperty Sınıfı Üye Bilgileri

Aşağıdaki örnekte öznitelikler kullanılarak yeni bir konfigürasyon bölüm sınıfı tanımlanması gösterilmiştir. Tanımlanan sınıf konfigurasyon.dll isimli Assembly içerisinde Konfigurasyon.Bolumler ad alanında bulunmaktadır.

C#

**public sealed** **class** OrnekBolum: ConfigurationSection {

[ConfigurationProperty("formBasligi", DefaultValue = "XXX",   
 IsRequired=**false**)]

**public** **string** FormBasligi {

**get** {

**return** (**string**)**this**["formBasligi"];

}

}

[ConfigurationProperty("sonGirisTarihi", IsRequired = **true**)]

**public** DateTime SonGirisTarihi {

**get** {

**return** (DateTime)**this**["sonGirisTarihi"];

}

**set** {

**this**["sonGirisTarihi"] = value;

}

}

}

VB.NET

**Public** **NotInheritable** **Class** OrnekBolum

**Inherits** ConfigurationSection

<ConfigurationProperty("formBasligi", DefaultValue:="XXX", \_

IsRequired:=**False**)> \_

**Public** **ReadOnly** **Property** FormBasligi() **As** **String**

**Get**

**Return** **CType**(**Me**("formBasligi"), **String**)

**End** **Get**

**End** **Property**

<ConfigurationProperty("sonGirisTarihi", IsRequired:=**True**)> \_

**Public** **Property** SonGirisTarihi() **As** DateTime

**Get**

**Return** **CType**(**Me**("sonGirisTarihi"), DateTime)

**End** **Get**

**Set**(**ByVal** Value **As** DateTime)

**Me**("sonGirisTarihi") = Value

**End** **Set**

**End** **Property**

**End** **Class**

Kod 332, Örnek Konfigürasyon Bölüm Sınıfı

Örnek sınıfta FormBasligi isimli özellik ön tanımlı değeri XXX olacak şekilde öznitelik ile donatılmıştır. Böylece bu özelliğin konfigürasyon dosyası içerisinde bulunmaması durumunda XXX değerinin kullanılması sağlanmış olmaktadır.

SonGirisTarihi isimli özelliğinin IsRequired özelliği true olarak belirlenmiş öznitelik ile donatılması, sonGirisTarihi isimli konfigürasyon özelliğinin konfigürasyon dosyasında bulunmasını zorunlu kılmaktadır.

Zorunluluk olmamasına rağmen, sınıf özellikleri ile konfigürasyon özellik isimlerinin aynı olması önerilmektedir. Genel prensip konfigürasyon özellik isimlerinin küçük harfle başlamasıdır.  
  
Sınıf tanımlamasının miras alınamaz şekilde yapılması önerilmektedir.

Tanımlanan sınıf ve bölüm bilgileri konfigürasyon dosyası içerisinde configSections bölümünde belirtilmelidir. Böylece, dosya içerisinde tanımlanan konfigürasyon bölümünün hangi sınıf ile yönetildiği konfigürasyon mimarisi tarafından anlaşılmış olmaktadır.

Tanımlanan sınıf ile kullanılabilecek örnek konfigürasyon dosyası aşağıda gösterilmiştir.

<configuration>

<**configSections**>

<**section** name="UygulamaAyarlari"   
 type="Konfigurasyon.Bolumler.OrnekBolum,Konfigurasyon"/>

</**configSections**>

<**UygulamaAyarlari**

sonGirisTarihi="1999-05-07 11:08:02"/>

</configuration>

Sınıf ve konfigürasyon dosyasında gerekli düzenlemelerin yapılması ardından ConfigurationManager.GetSection metodu kullanılarak bölüm nesnesi elde edilebilir.

C#

OrnekBolum bolum;

bolum = ConfigurationManager.GetSection("UygulamaAyarlari") **as** OrnekBolum;

**if** (bolum != **null**) {

Console.WriteLine(bolum.FormBasligi);

Console.WriteLine(bolum.SonGirisTarihi);

} **else** Console.WriteLine("Konfigürasyon dosyasında bölüm bulunamadı.");

VB.NET

**Dim** bolum **As** OrnekBolum

bolum = **TryCast**(ConfigurationManager.GetSection("UygulamaAyarlari"), \_  
 OrnekBolum)

**If** **Not** bolum **Is** **Nothing** **Then**

Console.WriteLine(bolum.FormBasligi)

Console.WriteLine(bolum.SonGirisTarihi)

**Else**

Console.WriteLine("Konfigürasyon dosyasında bölüm bulunamadı.")

**End** **If**

Kod 333, Programcı Tanımlı Konfigürasyon Bölümü Kullanımı

Kod çıktısı aşağıda gösterilmiştir.

XXX

07.05.1999 11:08:02

OrnekBolum.FormBasligi özelliği salt okunur tanımlanmış olmasına rağmen SonGirisTarihi özelliği okunabilir ve yazılabilir şekilde tanımlanmıştır. Böylece bu özelliğe atanan değerlerin tekrar konfigürasyon dosyasına yazılabilmesi sağlanmıştır.

C#

Configuration conf;

conf =ConfigurationManager.OpenExeConfiguration(ConfigurationUserLevel.None);

OrnekBolum bolum = conf.GetSection("UygulamaAyarlari") **as** OrnekBolum;

bolum.SonGirisTarihi = **new** DateTime(2009, 12, 5);

conf.SaveAs(@"c:\konfigurasyon.config");

VB.NET

**Dim** conf **As** Configuration

conf = ConfigurationManager.OpenExeConfiguration(ConfigurationUserLevel.None)

**Dim** bolum **As** OrnekBolum = **TryCast**(conf.GetSection("UygulamaAyarlari"), \_  
 OrnekBolum)

bolum.SonGirisTarihi = **New** DateTime(2009, 12, 5)

conf.SaveAs("c:\konfigurasyon.config")

Kod 334, Programcı Tanımlı Konfigürasyon Değerinin Kaydedilmesi

Kod çıktısından da görülebileceği gibi atanan değer konfigürasyon dosyasına yazılmıştır.

<configuration>

<configSections>

<section name="UygulamaAyarlari"

type="Konfigurasyon.Bolumler.OrnekBolum,Konfigurasyon"/>

</configSections>

<UygulamaAyarlari **sonGirisTarihi**="2009-12-05" />

</configuration>

Programcı Tanımlı Konfigürasyon Grupları

Soyut bir sınıf olan ConfigurationSectionGroup sınıfı miras alınarak yeni konfigürasyon bölüm grubu sınıfları tanımlanabilmesi mümkündür.

C#

**public** **sealed** **class** OrnekGrup: ConfigurationSectionGroup {

[ConfigurationProperty("ornekBolum")]

**public** OrnekBolum OrnekBolum {

**get** {

**return** (OrnekBolum)Sections["ornekBolum"];

}

}

}

VB.NET

**Public** **NotInheritable** **Class** OrnekGrup

**Inherits** ConfigurationSectionGroup

<ConfigurationProperty("ornekBolum")> \_

**Public** **ReadOnly** **Property** OrnekBolum() **As** OrnekBolum

**Get**

**Return** **CType**(Sections("ornekBolum"), OrnekBolum)

**End** **Get**

**End** **Property**

**End** **Class**

Kod 335, Örnek Konfigürasyon Bölüm Grubu Sınıfı

Konfigürasyon bölüm grubunun konfigürasyon dosyasında tanımlanması ve örnek kod parçası aşağıda gösterilmiştir.

<configuration>

<configSections>

<sectionGroup name ="uygulama.ayarlari"

type ="Konfigurasyon.Bolumler.OrnekGrup,Konfigurasyon">

<section name ="ornekBolum"

type="Konfigurasyon.Bolumler.OrnekBolum,Konfigurasyon"/>

</sectionGroup>

</configSections>

<uygulama.ayarlari>

<ornekBolum

formBasligi ="test"

sonGirisTarihi ="2009-01-01"/>

</uygulama.ayarlari>

</configuration>

C#

Configuration conf;

conf =ConfigurationManager.OpenExeConfiguration(ConfigurationUserLevel.None);

OrnekGrup grup = (OrnekGrup)conf.GetSectionGroup("uygulama.ayarlari");

Console.WriteLine(grup.OrnekBolum.FormBasligi);

VB.NET

**Dim** conf **As** Configuration

conf = ConfigurationManager.OpenExeConfiguration(ConfigurationUserLevel.None)

**Dim** grup **As** OrnekGrup = **CType**(conf.GetSectionGroup("uygulama.ayarlari"), \_  
 OrnekGrup)

Console.WriteLine(grup.OrnekBolum.FormBasligi)

Kod 336, Programcı Tanımlı Konfigürasyon Grubu Kullanımı

Kod çıktısı aşağıda gösterilmiştir.

test

Konfigürasyon bölüm grubu içerisinde istenilen sayıda konfigürasyon bölümü bulunabilir. Kod parçasında içerisinde bir adet konfigürasyon bölümü bulunan grup örneklenmiştir.

VS.NET ile Konfigürasyon Yönetimi

Teorik düzeyde anlatılan .NET konfigürasyon mimarisinin VS.NET ile görsel olarak gerçekleştirimi mümkündür. Görsel yapılan düzenlemeler sonucu, şu ana kadar aktarılan yöntemler kullanılarak, VS.NET tarafından gerekli kaynak kod ve konfigürasyon dosya içeriği otomatik olarak oluşturulmaktadır.

Ayarlar Dosyası

VS.NET ile konfigürasyon yönetimi yapabilmek için projeye ayarlar dosyasının (ing. Settings File) eklenmesi gerekmektedir.

Ayarlar dosyası Add new item … komutu verilerek eklenebilir.

Ekran Çıktısı 74, VS.NET Ayarlar Dosyası

Ayarlar dosyasının eklenmesi için izlenebilecek bir diğer yöntem de Project Options penceresi Settings sekmesinin kullanımıdır.

Projede ayarlar dosyasının bulunması durumunda bu sekme içerisinde ayar bilgileri görüntülenmektedir. Ayarlar dosyasının projede bulunmaması durumunda sekme tarafından ekleme yapılabilmesi için gerekli bağlantı gösterilmektedir.

Ekran Çıktısı 75, Ayarlar Sekmesi

Ayarlar sekmesi yardımıyla ayar dosyasının eklenmesi durumunda proje içerisinde Properties (VB.NET projesi için My Project) klasörüne ayar dosyası eklenecektir.

Properties ve My Project klasör içerikleri ön tanımlı olarak gösterilmemektedir. İçeriklerin gösterilmesi için VS.NET çözüm penceresinde Show All Files komutu verilmelidir.

Proje birden çok ayar dosyası içerebilir. Bu durumda hangi ayar dosyası düzenlenmek istiyorsa ilgili ayar dosyasına çift tıklamak yeterli olmaktadır.

Ayarlar Dosyasının Düzenlenmesi

Ayarlar sekmesi veya ayar dosyası üzerine çift tıklanarak açılan ayar editörü ekran çıktısı aşağıda gösterilmiştir.

Ekran Çıktısı 76, Ayarlar Editörü

Name sütunu ayar adını, Type sütunu ayar tipini belirtmektedir. Ayar tipinin açılır liste kutusundan seçilmesi mümkündür.

Scope sütunu ayarın kullanıcı veya uygulama bazında olduğunu belirtmek için kullanılmaktadır. Value sütununda belirtilen değer ayarın ön tanımlı değeri olmaktadır.

Seçili ayar satırının detaylı bilgileri VS.NET özellikler penceresinde görüntülenmektedir.

Ekran Çıktısı 77, VS.NET Özellikler Penceresi ve Seçili Ayar Bilgileri

Otomatik Kod Üretimi

Ayar dosyasının projeye eklenmesi ile, üretilecek kaynak kodların saklanacağı dosya da otomatik olarak projeye eklenmektedir. Örneğin Settings.settings isimli ayar dosyası için C# ve VB.NET projeleri için sırasıyla Settings.Designer.cs ve Settings.Designer.vb dosyaları bulunmaktadır.

Settings.settings isimli dosya içeriği Open with … komutu verilerek görüntülenebilir. Xml yapısında olan dosya, programcı tarafından ayar editörü ile yapılan düzenlemeleri saklamak amacıyla kullanılmaktadır. Settings.Designer.cs ve Settings.Designer.vb içeriği, bu dosyada bulunan bilgiler tarafından VS.NET tarafından otomatik olarak oluşturulmaktadır.

Settings.settings içerisinde sadece konfigürasyon tanımları ve yapısı bulunmaktadır. Konfigürasyon bölümleri ve değerleri önceki konuda aktartılan teorik bilgiler ışığında konfigürasyon dosyasına yazılmaktadır. Böylece programcının kod geliştirerek yaptığı işlem daha üst düzeyde VS.NET tarafından yapılmış olmaktadır.

Aşağıdaki ekran çıktısında Settings.settings isimli ayar dosya özellikleri gösterilmiştir.

Ekran Çıktısı 78, Ayarlar Dosyası Özellikleri

SettingsSingleFileGenerator, Microsoft tarafından sağlanan kod üreticisidir. Benzer yöntem, tasarımı VS.NET ile yapılabilen ve yapılan tasarım sonucunda otomatik kod üretilebilen her dosya tipi için geçerlidir. Örneğin veri kümeleri (ing. Datasets) tasarımı da VS.NET ile yapılabilmekte, yapılan tasarım sonucunda otomatik olarak kaynak kod üretilebilmektedir.

Ekran Çıktısı 79, Veri Kümesi

Veri kümeleri için kod üreticisinin MSDataSetGenerator olarak tanımlanmış olduğuna dikkat edilmelidir.

Detayları bu konunun dışında olmakla birlikte görsel tasarım ve görsel tasarım sonucu otomatik kod üretimi gün geçtikçe daha çok kullanılmaya başlanmaktadır. VS.NET, programcı tarafından da kod üreticilerinin geliştirilebilmesine destek vermektedir.

Ayarlar Sınıfı

VS.NET tarafından üretilen kaynak kod içeriğinin bir bölümü sadeleştirilerek aşağıda gösterilmiştir.

C#

**internal** **sealed** **partial** **class** Settings : ApplicationSettingsBase {

**private** **static** Settings defaultInstance =

(Settings)ApplicationSettingsBase.Synchronized(**new** Settings());

**public** **static** Settings Default {

**get** {

**return** defaultInstance;

}

}

**public** **string** FormBasligi {

**get** {

**return** ((**string**)(**this**["FormBasligi"]));

}

**set** {

**this**["FormBasligi"] = **value**;

}

}

VB.NET

**Partial** **Friend** **NotInheritable** **Class** MySettings

**Inherits** ApplicationSettingsBase

**Private** **Shared** defaultInstance **As** MySettings = \_  
 **CType**(ApplicationSettingsBase.Synchronized(**New** MySettings),MySettings)

**Public** **Shared** **ReadOnly** **Property** [**Default**]() **As** MySettings

**Get**

**Return** defaultInstance

**End** **Get**

**End** **Property**

**Public** **Property** FormBasligi() **As** **String**

**Get**

**Return** **CType**(**Me**("FormBasligi"),**String**)

**End** **Get**

**Set**

**Me**("FormBasligi") = value

**End** **Set**

**End** **Property**

**End** **Class**

Kod 337, VS.NET Ayarlar Sınıfı

Yapılan işlemin ApplicationSettingsBase sınıfının miras alınması olduğuna dikkat edilmelidir. Tanımlamanın kısmi (ing. Partial) olarak yapılmış olması programcı tarafından da sınıfa yeni üyeler eklenebilmesini sağlamaktadır.

Kullanım Bilgileri

VS.NET tarafından üretilen ayar sınıfın örnek kullanımı aşağıda gösterilmiştir.

C#

Console.WriteLine("Okunan değer: {0}", Settings.Default.FormBasligi);

Settings.Default.FormBasligi = "Kaydedilen değer";

Settings.Default.Save();

Console.WriteLine("Yeni değer kaydedildi ... {0}", Settings.Default.FormBasligi);

Settings.Default.FormBasligi = "yeni değer";

Console.WriteLine("Değiştirilmiş değer: {0}", Settings.Default.FormBasligi);

Settings.Default.Reload();

Console.WriteLine("Ayarlar tekrar yüklendi ...");

Console.WriteLine("Yüklenen değer ... {0}", Settings.Default.FormBasligi);

Console.WriteLine("Ayarlar sıfırlandı ...");

Settings.Default.Reset();

Console.WriteLine("Ön tanımlı değer: {0}", Settings.Default.FormBasligi);

VB.NET

Console.WriteLine("Okunan değer: {0}", My.Settings.FormBasligi)

My.Settings.FormBasligi = "Kaydedilen değer"

My.Settings.Save()

Console.WriteLine("Yeni değer kaydedildi ... {0}", My.Settings.FormBasligi)

My.Settings.FormBasligi = "yeni değer"

Console.WriteLine("Değiştirilmiş değer: {0}", My.Settings.FormBasligi)

My.Settings.Reload()

Console.WriteLine("Ayarlar tekrar yüklendi ...")

Console.WriteLine("Yüklenen değer ... {0}", My.Settings.FormBasligi)

Console.WriteLine("Ayarlar sıfırlandı ...")

My.Settings.Reset()

Console.WriteLine("Ön tanımlı değer: {0}", My.Settings.FormBasligi)

Kod 338, VS.NET Tarafından Üretilen Ayar Sınıfı Kullanımı

Kod çıktısı aşağıda gösterilmiştir.

Okunan değer: Ön tanımlı değer

Yeni değer kaydedildi ... Kaydedilen değer

Değiştirilmiş değer: yeni değer

Ayarlar tekrar yüklendi ...

Yüklenen değer ... Kaydedilen değer

Ayarlar sıfırlandı ...

Ön tanımlı değer: Ön tanımlı değer

Ayar Olayları

Uygulama ayarları değiştirildiğinde veya ayarlar kaydedilmeden önce programcı tarafından bazı işlemler yapılmak istenebilir. Bu durumda kısmi ayar sınıfının tanımlanması ve ApplicationSettingsBase sınıfı delegelerini kullanmak yeterlidir.

Ayar editöründe View Code komutu verilmesi ile aynı isimli kısmi sınıf otomatik olarak VS.NET tarafından oluşturulmaktadır.

Aşağıda ayar olaylarının tanımlanması gösterilmiştir.

C#

**internal** **sealed** **partial** **class** Settings {

**public** Settings() {

**this**.SettingChanging += Settings\_SettingChanging;

**this**.SettingsSaving += Settings\_SettingsSaving;

**this**.SettingsLoaded += Settings\_SettingsLoaded;

**this**.PropertyChanged += Settings\_PropertyChanged;

}

**void** Settings\_PropertyChanged(**object** sender, PropertyChangedEventArgs e) {

Console.WriteLine("{0} değeri değişti ...", e.PropertyName);

}

**void** Settings\_SettingsSaving(**object** sender, CancelEventArgs e) {

Console.WriteLine("Ayarlar kaydediliyor ...");

}

**void** Settings\_SettingChanging(**object** sender, SettingChangingEventArgs e) {

Console.WriteLine("-----Ayar değeri değişmek üzere ----");

Console.WriteLine("Ayar adı: {0}", e.SettingName);

Console.WriteLine("Yeni değer: {0}", e.NewValue);

Console.WriteLine("-------------------------------------");

}

**void** Settings\_SettingsLoaded(**object** sender, SettingsLoadedEventArgs e) {

Console.WriteLine("Ayarlar yüklendi ...");

}

}

VB.NET

**Partial** **Friend** **NotInheritable** **Class** MySettings

**Public** **Sub** **New**()

**AddHandler** **Me**.SettingChanging, **AddressOf** Settings\_SettingChanging

**AddHandler** **Me**.SettingsSaving, **AddressOf** Settings\_SettingsSaving

**AddHandler** **Me**.SettingsLoaded, **AddressOf** Settings\_SettingsLoaded

**AddHandler** **Me**.PropertyChanged, **AddressOf** Settings\_PropertyChanged

**End** **Sub**

**Sub** Settings\_PropertyChanged(**ByVal** sender **As** **Object**, \_  
 **ByVal** e **As** PropertyChangedEventArgs)

Console.WriteLine("{0} değeri değişti ...", e.PropertyName)

**End** **Sub**

**Sub** Settings\_SettingsSaving(**ByVal** sender **As** **Object**, \_  
 **ByVal** e **As** CancelEventArgs)

Console.WriteLine("Ayarlar kaydediliyor ...")

**End** **Sub**

**Sub** Settings\_SettingChanging(**ByVal** sender **As** **Object**,   
 **ByVal** e **As** SettingChangingEventArgs)

Console.WriteLine("-----Ayar değeri değişmek üzere ----")

Console.WriteLine("Ayar adı: {0}", e.SettingName)

Console.WriteLine("Yeni değer: {0}", e.NewValue)

Console.WriteLine("-------------------------------------")

**End** **Sub**

**Sub** Settings\_SettingsLoaded(**ByVal** sender **As** **Object**, \_  
 **ByVal** e **As** SettingsLoadedEventArgs)

Console.WriteLine("Ayarlar yüklendi ...")

**End** **Sub**

**End** **Class**

Kod 339, Ayar Olayları Kullanımı

Aşağıdaki kod örneğinde ayarların değiştirilmesi, yeniden yüklenmesi ve kaydedilmesi gösterilmiştir.

C#

Settings.Default.FormBasligi = "Yeni başlık";

Settings.Default.Save();

Settings.Default.Reload();

VB.NET

My.Settings.FormBasligi = "Yeni başlık"

My.Settings.Save()

My.Settings.Reload()

Kod 340, Ayar Olaylarının Tetiklenmesi

Yapılan işlemler sonucu üretilen çıktı aşağıda gösterilmiştir.

-----Ayar değeri değişmek üzere ----

Ayar adı: FormBasligi

Yeni değer: Yeni başlık

-------------------------------------

FormBasligi değeri değişti ...

Ayarlar kaydediliyor ...

FormBasligi değeri değişti ...

Ayar Sağlayıcıları

Bilindiği gibi uygulama ayar değerleri uygulama konfigürasyon dosyasından okunmakta ve bu dosya içerisinde saklanmaktadır. Ayar sağlayıcıları (ing. Setting Providers), ayarların istenilen ortamda saklanabilmesi için pratik bir çözüm sunmaktadır.

Ayar sağlayıcısını tüm ayarlar için veya her bir için uygulama ayarı için ayrı ayrı belirtmek mümkündür. Böylece, bazı ayarlar konfigürasyon dosyasında saklanırken, istenilen ayarlar geliştirilen ayar sağlayıcı sınıfları yardımıyla farklı ortamlarda saklanabilir.

Ayar Sağlayıcı Sınıfları

Sağlayıcı sınıfları, System.Configuration.Provider.ProviderBase soyut sınıfını miras alan sınıflardır. Detayları bu konunun dışında olmakla birlikte bu sınıf, sağlayıcı bazlı tasarım modelinin (ing. Provider Design Pattern) uygulanmasında kullanılan temel sınıftır.

VS.NET ayar editörü ile herhangi bir ayar satırı seçilerek VS.NET özellikler penceresinde Provider değerine sağlayıcı sınıfın girilmesi ile seçili ayar için ayar sağlayıcısı belirtilebilmektedir.

Ekran Çıktısı 80, Uygulama Ayar Sağlayıcı Özelliği

Ayar sağlayıcısını belirtmek için öznitelikler kullanılmaktadır. VS.NET özellikler penceresi ile yapılan işlem sonucu FormBasligi isimli özellik SettingsProviderAttribute özniteliği ile donatılmaktadır.

C#

[SettingsProviderAttribute(**typeof**(Saglayicilar.VeriTabaniAyarSaglayicisi))]

**public** **string** FormBasligi {

**get** {

**return** ((**string**)(**this**["FormBasligi"]));

}

**set** {

**this**["FormBasligi"] = **value**;

}

}

VB.NET

<SettingsProviderAttribute(**GetType**(Saglayicilar.VeriTabaniAyarSaglayicisi))>

**Public** **Property** FormBasligi() **As** **String**

**Get**

**Return** **CType**(**Me**("FormBasligi"),**String**)

**End** **Get**

**Set**

**Me**("FormBasligi") = value

**End** **Set**

**End** **Property**

Kod 341, Ayar Özelliklerinin Ayar Sağlayıcı Özniteliği

Ayar sınıfının SettingsProviderAttribute özniteliği ile donatılması tüm ayarlar için belirtilen ayar sağlayıcı sınıfının geçerli olmasını sağlamaktadır.

C#

[SettingsProviderAttribute(**typeof**(Saglayicilar.VeriTabaniAyarSaglayicisi))]

**internal** **sealed** **partial** **class** Settings {

. . .

}

VB.NET

<SettingsProviderAttribute(**GetType**(Saglayicilar.VeriTabaniAyarSaglayicisi))>

**Partial** **Friend** **NotInheritable** **Class** MySettings

. . .

**End** **Class**

Kod 342, Ayar Sınıfının Öznitelik ile Donatılması

Ayar sınıfının öznitelik ile donatılması için VS.NET tarafından üretilen dosya içerisindeki (Settings.Designer.cs ve Settings.Designer.vb) sınıfın kullanılması kod üretiminin tekrar yapılması ile bu tanımın kaybolmasını beraberinde getirmektedir. Ayar editöründe View Code komutu verilerek yeni bir kaynak dosya oluşturulması ve bu dosya içerisindeki sınıfın öznitelik ile donatılması önerilmektedir.

Programcı tarafından ayar sağlayıcısının belirtilmediği durumlarda LocalFileSettingsProvider sınıfı ön tanımlı ayar sağlayıcı sınıfı olarak kullanılmaktadır.

Ayar Sağlayıcı Sınıfın Geliştirilmesi

ProviderBase sınıfını miras alan SettingsProvider sınıfı programcıların yeni ayar sağlayıcı sınıfları geliştirmek için miras alması gereken ata sınıftır.

Aşağıda şablon olarak kullanılabilecek örnek sınıf gösterilmiştir.

C#

**class** VeriTabaniAyarSaglayicisi: SettingsProvider {

**public** **override** **string** ApplicationName {

**get** { **return** "Uygulama Adı"; }

**set** { }

}

**public** **override** SettingsPropertyValueCollection   
 GetPropertyValues(SettingsContext context,   
 SettingsPropertyCollection collection) {

SettingsPropertyValueCollection degerler

degerler = **new** SettingsPropertyValueCollection();

**foreach** (SettingsProperty ozellik **in** collection) {

SettingsPropertyValue deger = **new** SettingsPropertyValue(ozellik);

deger.SerializedValue = Oku(ozellik.Name);

degerler.Add(deger);

}

**return** degerler;

}

**private** **object** Oku(**string** ad) {

// ad isimli özellik değeri bu bölümde istenilen ortamdan okunabilir.

}

**public** **override** **void** SetPropertyValues(SettingsContext context,   
 SettingsPropertyValueCollection collection) {

**foreach** (SettingsPropertyValue ozellikDegeri **in** collection) {

**if** (!ozellikDegeri.IsDirty || (ozellikDegeri.SerializedValue == **null**)){

**continue**;

}

Kaydet(ozellikDegeri.Name, ozellikDegeri.SerializedValue);

}

}

**private** **void** Kaydet(**string** ad, **object** deger) {

// ad isimli özelliğe atanmış olan deger bu bölümde istenilen ortama  
 // kaydedilebilir.

}

**public** **override** **void** Initialize(**string** name,   
 System.Collections.Specialized.NameValueCollection config){

**base**.Initialize(ApplicationName, config);

}

}

VB.NET

**Class** VeriTabaniAyarSaglayicisi

**Inherits** SettingsProvider

**Public** **Overrides** **Property** ApplicationName() **As** **String**

**Get**

**Return** "Uygulama Adı"

**End** **Get**

**Set**(**ByVal** value **As** **String**)

**End** **Set**

**End** **Property**

**Public** **Overrides** **Function** GetPropertyValues(**ByVal** context **As** \_  
 SettingsContext, \_  
 **ByVal** collection **As** SettingsPropertyCollection) \_  
 **As** SettingsPropertyValueCollection

**Dim** degerler **As** SettingsPropertyValueCollection

degerler = **New** SettingsPropertyValueCollection()

**For** **Each** ozellik **As** SettingsProperty **In** collection

**Dim** deger **As** SettingsPropertyValue = **New** SettingsPropertyValue(ozellik)

deger.SerializedValue = Oku(ozellik.Name)

degerler.Add(deger)

**Next**

**Return** degerler

**End** **Function**

**Private** **Function** Oku(**ByVal** ad **As** **String**) **As** **Object**

' ad isimli özellik değeri bu bölümde istenilen ortamdan okunabilir.

**End** **Function**

**Public** **Overrides** **Sub** SetPropertyValues(**ByVal** context **As** SettingsContext,  
 **ByVal** collection **As** SettingsPropertyValueCollection)

**For** **Each** ozellikDegeri **As** SettingsPropertyValue **In** collection

**If** **Not** ozellikDegeri.IsDirty **Or** \_  
 ozellikDegeri.SerializedValue **Is** **Nothing** **Then**

**Continue** **For**

**End** **If**

Kaydet(ozellikDegeri.Name, ozellikDegeri.SerializedValue)

**Next**

**End** **Sub**

**Private** **Sub** Kaydet(**ByVal** ad **As** **String**, **ByVal** deger **As** **Object**)

' ad isimli özelliğe atanmış olan deger bu bölümde istenilen ortama  
 ' kaydedilebilir.

**End** **Sub**

**Public** **Overrides** **Sub** Initialize(**ByVal** name **As** **String**, \_  
 **ByVal** config **As** System.Collections.Specialized.NameValueCollection)

**MyBase**.Initialize(ApplicationName, config)

**End** **Sub**

**End** **Class**

Kod 343, Örnek Ayar Sağlayıcı Sınıfı

IApplicationSettingsProvider arabirim gerçekleştiriminin yapılması ile yukarıdaki sınıfın genişletilmesi mümkündür.

Konfigürasyon Güvenliği

Konfigürasyon bilgilerinin güvenliğinin sağlanması güvenli yazılımlar geliştirebilmenin önemli adımlarındandır.

Konfigürasyon bilgilerinin güvenliğini sağlamak amacıyla iki temel yöntem izlenebilir.

Kitabın Veri Güvenliği ve Elektronik İmza bölümünün konuya başlamadan önce gözden geçirilmesi önerilmektedir.

Konfigürasyon Ayar Sağlayıcıları

Programcı tarafından geliştirilen ayar sağlayıcıları kullanılarak konfigürasyon bilgilerinin şifrelenerek kaydedilmesi mümkündür. Bu yöntemin uygulanmasında kitabın önceki bölümlerinde aktarılan veri koruma sınıf ve metotları kullanılabilir.

Korunumlu Konfigürasyon Sağlayıcıları

Konfigürasyon bilgilerinin korunabilmesi amacıyla .NET konfigürasyon mimarisi tarafından çeşitli yöntemler sunulmaktadır.

Sunulan yöntemler, verinin korunması amacıyla .NET sınıf kütüphanesi içerisinde bulunan hazır veya programcı tarafından geliştirilen korunumlu konfigürasyon sağlayıcı (ing. Protected Configuration Provider) sınıflarının kullanımı esasına dayalıdır.

DPAPI Sağlayıcısı

Windows veri koruma API (DPAPI) sağlayıcısı konfigürasyon bilgisinin korunması amacıyla kullanılabilir.

Korunmamış bir konfigürasyon dosyası örneği aşağıda gösterilmiştir.

<configuration>  
 <**appSettings**>

<add key ="Ayar1" value="Ayar1Deger"/>

<add key="Ayar2" value ="Ayar2Deger" />

</**appSettings**>

</configuration>

DPAPI kullanılarak aşağıdaki şekilde konfigürasyon dosyasının korunması mümkündür.

C#

**private** **static** **void** Sifrele() {

Configuration conf;

conf=ConfigurationManager.OpenExeConfiguration(ConfigurationUserLevel.None);

// conf.GetSection kullanılarak istenilen konfigürasyon bölümü   
 // kullanılabilir.

ConfigurationSection sec = conf.AppSettings;

**string** saglayici = "DataProtectionConfigurationProvider";

// Konfigürasyon bölümü korunmamış ise işleme başla

**if** (!sec.SectionInformation.IsProtected) {

// Kullanılmak istenen sağlayıcıyı parametre olarak ilet.

sec.SectionInformation.ProtectSection(saglayici);

sec.SectionInformation.ForceSave = **true**;

}

**else**

Console.WriteLine("{0} zaten korunmaktadır.",sec.SectionInformation.Name);

conf.Save();

}

**private** **static** **void** Coz() {

Configuration conf;

conf=ConfigurationManager.OpenExeConfiguration(ConfigurationUserLevel.None);

ConfigurationSection sec = conf.AppSettings;

**if** (sec.SectionInformation.IsProtected) {

sec.SectionInformation.UnprotectSection();

sec.SectionInformation.ForceSave = **true**;

}

**else**

Console.WriteLine("{0} zaten korunumludur.", sec.SectionInformation.Name);

conf.Save();

}

VB.NET

**Shared** **Sub** Sifrele()

**Dim** conf **As** Configuration

conf=ConfigurationManager.OpenExeConfiguration(ConfigurationUserLevel.None)

' conf.GetSection kullanılarak istenilen konfigürasyon bölümü   
 ' kullanılabilir.

**Dim** sec **As** ConfigurationSection = conf.AppSettings

**Dim** saglayici **As String =** "DataProtectionConfigurationProvider"

' Konfigürasyon bölümü korunmamış ise işleme başla

**If** **Not** sec.SectionInformation.IsProtected **Then**

' Kullanılmak istenen sağlayıcıyı parametre olarak ilet.

sec.SectionInformation.ProtectSection(saglayici)

sec.SectionInformation.ForceSave = **True**

**Else**

Console.WriteLine("{0} zaten korunmaktadır.",sec.SectionInformation.Name)

**End** **If**

conf.Save()

**End** **Sub**

**Shared** **Sub** Coz()

**Dim** conf **As** Configuration

conf=ConfigurationManager.OpenExeConfiguration(ConfigurationUserLevel.None)

**Dim** sec **As** ConfigurationSection = conf.AppSettings

**If** sec.SectionInformation.IsProtected **Then**

sec.SectionInformation.UnprotectSection()

sec.SectionInformation.ForceSave = **True**

**Else**

Console.WriteLine("{0} zaten korunumludur.", sec.SectionInformation.Name)

**End** **If**

conf.Save()

**End** **Sub**

Kod 344, DPAPI Kullanılarak Konfigürasyon Bilgilerinin Korunması

Veri koruması yapılan konfigürasyon bölümü aşağıda gösterilmiştir.

<appSettings configProtectionProvider="DataProtectionConfiguration...">

<EncryptedData>

<CipherData>

<CipherValue>AQAAANCMnd8BFdERjHoAwE . . .</CipherValue>

</CipherData>

</EncryptedData>

</appSettings>

Uygulama tarafından konfigürasyon bilgilerine erişimde herhangi bir ek işlem yapılması gerekmemektedir. .NET konfigürasyon mimarisi tarafından configProtectionProvider öznitelik değerine göre otomatik olarak şifre çözme işlemi yapılmaktadır.

Bilindiği gibi DPAPI şifreleme anahtarı gerektirmemektedir ve her bilgisayarda farklı bir anahtar otomatik olarak üretilmektedir. Geliştirme yapılan bilgisayarda verinin DPAPI ile korunması ve yazılımın bir başka bilgisayara bu şekliyle yüklenmesi durumunda verinin geçersizleşeceği unutulmamalıdır. Web uygulamalarında bu problemi elimine etmek için uygulamanın sunucuya yüklenmesi aşamasında veriyi sunucu üzerinde şifrelemek gerekmektedir. Pratik bir yöntem sifrele.aspx isimli sayfanın uzaktan istenmesi ve bu sayfanın konfigürasyon bilgilerini şifrelemesidir.  
  
Kullanılabilecek bir diğer yöntem sunucuya yükleme tamamlandıktan sonra aşağıdaki komutu çalıştırmaktır.  
  
aspnet\_regiis -pe "Konfigürasyon Bölümü" -app "/Uygulama" -prov "Saglayici Adı"

Programcı Tanımlı Sağlayıcılar

Programcı tarafından geliştirilen korunumlu konfigürasyon sağlayıcı sınıfları ile istenilen şekilde verinin korunması mümkündür. Bunun için öncelikle kullanılan sağlayıcı uygulama konfigürasyon dosyası içerisinde tanımlanmalıdır.

<configuration>

<configProtectedData >

<providers>

<add name="OzelKoruyucu"   
 type="Koruyucular.KonfigurasyonKoruyucusu,Assembly"   
 ozelDeger1="xxx"   
 ozelDeger2="yyy"   
 />

</providers>

</configProtectedData >

</configuration>

Korunumlu konfigürasyon sağlayıcı sınıfı geliştirmek için System.Configuration.ProtectedConfigurationProvider sınıfının miras alınması gerekmektedir.

Aşağıda şablon olarak kullanılabilecek sınıf örneği gösterilmiştir.

C#

**class** KonfigurasyonKoruyucusu : ProtectedConfigurationProvider {

**private** **string** sifrele(**string** duzMetin) {

// Burada istenilen yöntem kullanılabilir.

}

**private** **string** coz(**string** sifreliMetin) {

// Burada istenilen yöntem kullanılabilir.

}

**public** **override** XmlNode Decrypt(XmlNode encryptedNode) {

XmlDocument xmlDoc = **new** XmlDocument();

xmlDoc.PreserveWhitespace = **true**;

**string** xml = **string**.Format("<EncryptedData>{0}</EncryptedData>",   
 coz(encryptedNode.InnerText));

xmlDoc.LoadXml(xml);

**return** xmlDoc.DocumentElement;

}

**public** **override** XmlNode Encrypt(XmlNode node) {

XmlDocument xmlDoc = **new** XmlDocument();

xmlDoc.PreserveWhitespace = **true**;

**string** xml = **string**.Format("<EncryptedData>{0}</EncryptedData>",  
 sifrele(node.InnerText));

xmlDoc.LoadXml(xml);

**return** xmlDoc.DocumentElement;

}

**public** **override** **void** Initialize(**string** name, NameValueCollection config) {

// Sağlayıcıya iletilen değerler burada kullanılabilir.

**string** deger1 = config["ozelDeger1"];

}

}

VB.NET

**Class** KonfigurasyonKoruyucusu

**Inherits** ProtectedConfigurationProvider

**Private** **Function** sifrele(**ByVal** duzMetin **As** **String**) **As** **String**

' Burada istenilen yöntem kullanılabilir.

**End** **Function**

**Private** **Function** coz(**ByVal** sifreliMetin **As** **String**) **As** **String**

' Burada istenilen yöntem kullanılabilir.

**End** **Function**

**Public** **Overrides** **Function** Decrypt(**ByVal** encryptedNode **As** XmlNode) **As** XmlNode

**Dim** xmlDoc **As** XmlDocument = **New** XmlDocument()

xmlDoc.PreserveWhitespace = **True**

**Dim** xml **As** **String** = **String**.Format("<EncryptedData>{0}</EncryptedData>", \_  
 coz(encryptedNode.InnerText))

xmlDoc.LoadXml(xml)

**Return** xmlDoc.DocumentElement

**End** **Function**

**Public** **Overrides** **Function** Encrypt(**ByVal** node **As** XmlNode) **As** XmlNode

**Dim** xmlDoc **As** XmlDocument = **New** XmlDocument

xmlDoc.PreserveWhitespace = **True**

**Dim** xml **As** **String** = **String**.Format("<EncryptedData>{0}</EncryptedData>", \_  
 sifrele(node.InnerText))

xmlDoc.LoadXml(xml)

**Return** xmlDoc.DocumentElement

**End** **Function**

**Public** **Overrides** **Sub** Initialize(**ByVal** name **As** **String**, \_  
 **ByVal** config **As** NameValueCollection)

**Dim** deger1 **As** **String** = config("ozelDeger1")

**'** Sağlayıcıya iletilen değerler burada kullanılabilir.

**End** **Sub**

**End** **Class**

Kod 345, Programcı Tanımlı Korunumlu Konfigürasyon Sağlayıcısı

Oluşturulan sağlayıcı kullanımı bir önceki konuda aktarılan yöntemin aynısıdır. saglayici isimli yerel değişkene DataProtectionConfigurationProvider değeri yerine OzelKoruyucu değerinin atanması yeterli olmaktadır.

Araştırma yapıldığı zaman ancak bilgi artırılabilir; bilgi artırıldığında ancak istek samimi olabilir; istek samimi olduğunda ancak akıl ıslah edilebilir; akıl ıslah edildiğinde ancak özel yaşam iyileştirilebilir; özel yaşam iyileştirildiğinde ancak aile yapısı düzeltilebilir. Aile yapısı düzeltildiğinde ancak devlet düzen içinde yönetilebilir.

Konfüçyus

Bölüm

23

Windows Hizmet Uygulamaları

P

rogramlama sırasında bazı işlemlerin arka planda çalışması istenebilir. Arka planda çalışan uygulama, uzun süren veya belirli zaman aralıkları ile tekrarlanması gereken işlemleri herhangi bir kullanıcı ara yüzü olmadan yapabilir.

Bu bölümde Windows hizmet uygulamaları ve bu uygulamaların .NET kullanılarak geliştirilmesi hakkında bilgiler sunulacaktır.

Hizmet Uygulamaları Hakkında

Windows hizmet uygulamaları (ing. Windows Service Applications), işletim sisteminin aktif olması ile birlikte arka planda çalışabilen ve Windows hizmet uygulama mimarisi (ing. Windows Service Application Architecture) kurallarına göre geliştirilmiş uygulama türüdür.

Hizmet uygulamalarını konsol veya Windows Forms uygulamalarından ayıran en önemli özellik, hizmet uygulamalarının çalışabilmesi için kullanıcının bilgisayara giriş yapmasının zorunlu olmamasıdır. Diğer uygulama türlerinde, uygulamalar sisteme giriş yapmış kullanıcı altında çalışırken hizmet uygulamaları işletim sisteminin aktif olması ile birlikte çalışabilmektedir.

Hizmet uygulamalarına örnek olarak ilişkisel veritabanı yönetim sistemleri verilebilir. Veritabanı yönetim sistemi yazılımı hizmet uygulaması olarak arka planda çalışmakta ve bağlantı isteklerini dinlemektedir. Böylece veritabanı yönetim sisteminin çalışabilmesi için sunucuya bir kullanıcının giriş yapması ve uygulamayı çalıştırması gerekmemektedir.

Hizmet uygulamalarına verilebilecek bir diğer örnek de Microsoft internet bilgi sunucusu (Microsoft Internet Information Server – IIS) yazılımıdır. Uygulama sunucusu (ing. Application Server) olarak çalışan IIS, TCP port 80 üzerinde kullanıcılardan gelen istekleri dinlemekte ve http (Hyper Text Transfer Protocol) ile isteklere yanıt vermektedir.

Kullanım Alanları

Hizmet uygulamaları programcılar tarafından genellikle teorik bilgi eksikliğinden dolayı çok fazla kullanılıyor olmamasına rağmen, sağladıkları kazanımlar ile bir çok problemin çözümünde sağlam bir alt yapı sunabilmektedir.

Örneğin web uygulamaları uzun süren veya belirli zaman aralıkları ile tekrarlanması gereken işlemlerin yapılması için uygun değildir. Bu tür ihtiyaçların web sayfası zaman aşımı süresinin (ing. Page Timeout) artırılarak çözülmesi gibi yaklaşımlar bulunsa da, güvenlik ve performans kriterleri değerlendirildiğinde sağlam çözümler olmaktan uzaktadırlar. Bu tür ihtiyaçların hizmet uygulamaları ile karşılanması sistem güvenliğini ve performansını önemli ölçüde artırabilmektedir.

Hizmet Uygulamaları Özellikleri

Hizmet uygulamaları diğer uygulama türlerinden farklı bir mimariye sahiptir. Bu uygulamaların çalıştırılması, durdurulması ve güvenlik özelliklerinin belirlenmesi gibi temel konular hakkında bu konu başlığında bilgiler verilecektir.

Hizmet uygulamalarının yönetimi için Windows tarafından hizmet kontrol yöneticisi (ing. Service Control Manager - SCM) bileşeni sunulmaktadır. SCM, işletim sisteminde yüklü hizmet uygulamalarının yönetiminin yapılabilmesi amacıyla temel altyapıyı sağlamaktadır. SCM tarafından sağlanan fonksiyonlar ile yüklü hizmetlerin listelenmesi, hizmet uygulamalarının başlatılması, durdurulması gibi işlemler yapılabilmektedir.

Hizmet ve Hizmet Uygulaması

Hizmet ve hizmet uygulaması birbirlerine karıştırılabilen kavramlardır. Hizmet uygulaması bir veya birden fazla hizmet içerebilir. Örneğin bir çalışabilir dosya (.exe) içerisinde birden fazla hizmet bulunabilir. Böylece aynı proses içerisinde birden çok hizmetin çalışabilmesi sağlanmış olmaktadır.

Genel Hizmet Özellikleri

Her hizmet, hizmeti işletim sisteminde tekil olarak ifade eden hizmet adı (ing. Service Name) değerine sahiptir. Hizmet adı değeri SCM tarafından yönetilen hizmet bilgilerine erişmek için kullanılmaktadır.

Hizmet görünüm adı (ing. Service Display Name), hizmetin adını daha açıklayıcı olarak ifade etmektedir. İsteğe bağlı olarak hizmetin daha uzun bir açıklama değeri (ing. Service Description) de olabilmektedir.

Başlatma Tipi

Hizmet uygulamaları işletim sisteminin aktif olması ile birlikte veya kullanıcı tarafından başlatılabilmektedir. Başlama tipi (ing. Startup Type) olarak ifade edilebilecek bu kavram aşağıdaki değerleri alabilmektedir.

| Başlama Tipi | Açıklama |
| --- | --- |
| Atomatik (ing. Automatic) | Hizmetin işletim sistemi ile birlikte otomatik olarak çalışacağını belirten değerdir. |
| Elle (ing. Manual) | Hizmetin kullanıcı tarafından başlatılacağını belirten değerdir. |
| Devre Dışı (ing. Disabled) | Hizmetin kullanılamaz durumunda olduğunu belirten değerdir. |

Tablo 133, Hizmet Uygulamaları Başlama Tipi Değer ve Açıklamaları

Hizmet Hesabı

Çalışan her hizmetin, yetki ve güvenlik kriterlerin belirleyen bir işletim sistemi kullanıcısı ile ilişkilendirilmesi gerekmektedir. Hizmet hesabı (ing. Service Account) olarak adlandırılan bu kavram, diğer uygulama türlerinde olduğu gibi hizmetin altında çalıştığı kullanıcı adıdır. Buna rağmen hizmetin çalışabilmesi için, seçilen kullanıcının işletim sistemine giriş yapmış olma zorunluluğunun bulunmadığına dikkat edilmelidir.

LocalSystem isimli hizmet hesabı SCM tarafından yönetilen özel bir kullanıcı hesabıdır. Bu hesap, herhangi bir kullanıcı adı ve şifresi belirtilmeden hizmetin yönetici kullanıcı yetkileriyle çalışabilmesini sağlamaktadır.

Hizmet Durumu

Bir hizmet üç temel durumda olabilir.

* Başlatılmış (ing. Started)
* Duraklatılmış (ing. Paused)
* Durdurulmuş (ing. Stopped)

Hizmete SCM tarafından başlat komutunun gönderilmesi ve bu komutun hizmet tarafından anlaşılması ile hizmet durumu Başlatılmış olmaktadır. Başlatılmış hizmet kavramından sağladığı işlevleri yapan aktif hizmet anlaşılmalıdır.

Duraklatılmış hizmet, SCM tarafından durakla komutunun hizmete iletilmesi ile, hizmetin yaptığı işlemi askıya almasını ifade etmektedir. Duraklatılmış hizmet kavramından aktif olan ancak sağladığı işlevi yapmayan hizmet anlaşılmalıdır.

Durdurulmuş hizmet, SCM tarafından iletilen durdur komutu ile hizmetin çalışmayı durdurmuş olmasını ifade etmektedir. Durdurulmuş hizmet kavramından çalışmayan hizmet anlaşılmalıdır.

Bazı hizmetler duraklatılmayı veya durdurulmayı desteklemeyebilir. Örneğin olay yönetimini yapan Event Log isimli hizmet durdurulamaz özelliğe sahiptir.

Hizmet Bağımlılıkları

Her hizmet, bir başka hizmet veya hizmetlere bağımlı olabilir. Hizmet bağımlılıkları (ing. Service Dependency) ile hedeflenen, hizmetin diğer hizmetler tarafından sunulan işlevlere eriştiğinin belirtilmesidir. Böylece bir hizmet başlamadan önce bağımlı olduğu hizmetlerin çalışır durumda olmaları sağlanabilmektedir.

Hizmet Yönetim Uygulaması

Hizmetlerin yönetiminin yapılabilmesi amacıyla farklı yazılımlar kullanılabilir. Windows işletim sistemi ile birlikte gelen ve Microsoft yönetim konsol uygulaması olan hizmet yönetim uygulaması kontrol panel | yönetimsel araçlar klasöründe bulunan kısa yol yardımıyla çalıştırılabilir.

Ekran Çıktısı 81, Windows Hizmet Yöneticisi

Hizmet yönetim uygulaması yardımıyla hizmetlerin başlatılması, durdurulması, duraklatılması ve yukarıda aktarılan özelliklerinin düzenlenmesi mümkündür.

Hizmetlerin yönetimi için kullanılabilecek bir diğer uygulama da sc.exe isimli komut satırı uygulamasıdır. Sc.exe ile hizmetlerin yüklenmesi, sistemden kaldırılması gibi daha ileri seviye işlemler de yapılabilmektedir.

.NET Hizmet Uygulamaları ve Sınıfı

System.ServiceProcess.ServiceBase sınıfı Windows hizmet uygulamaları geliştirmek için kullanılan temel sınıftır. Sınıf tarafından sağlanan işlevler ile bir önceki konuda teorik düzeyde aktarılan hizmet özelliklerinin belirlenmesi mümkündür.

ServiceBase sınıfı System.ServiceProcess.dll isimli Assembly içerisinde bulunmaktadır ve VS.NET hizmet uygulaması şablonu ile projenin bu dosyaya referans vermesi sağlanmaktadır. Hizmet uygulamaları geliştirilirken VS.NET kullanılmaması durumunda programcı tarafından referansın verilmesi gerekmektedir.

Hizmet Uygulamaları

Hizmet uygulamasını VS.NET proje şablonları yardımıyla kolaylıkla oluşturmak mümkündür. Bunun için yeni proje komutunun verilmesi ve Windows Service elemanının seçilmesi mümkündür.

Ekran Çıktısı 82, VS.NET ile Windows Hizmet Projesi Eklenmesi

Bilindiği gibi hizmet uygulaması içerisinde bir veya daha fazla hizmet bulunabilmektedir. Windows hizmet projesi eklenmesiyle, uygulama içerisindeki hizmeti ifade eden sınıf da VS.NET tarafından oluşturulmaktadır.

Proje şablonu ile oluşturulan Main metodu aşağıda gösterilmiştir.

C#

**static** **void** Main() {

ServiceBase[] ServicesToRun;

ServicesToRun = **new** ServiceBase[] { **new** Service1() };

ServiceBase.Run(ServicesToRun);

}

VB.NET

**Shared** **Sub** Main()

**Dim** ServicesToRun() **As** System.ServiceProcess.ServiceBase

ServicesToRun = **New** System.ServiceProcess.ServiceBase() { **New** Service1() }

System.ServiceProcess.ServiceBase.Run(ServicesToRun)

**End** **Sub**

Kod 346, Windows Hizmet Uygulaması Main Metodu

C# projeleri için Main metodu program.cs kaynak kod dosyasında, VB.NET projelerinde hizmet sınıfı içerisinde bulunmaktadır.

ServiceBase.Run metodu hizmet sınıfları dizisini parametre olarak almakta ve hizmetleri çalıştırmaktadır. Uygulama içerisinde birden çok hizmetin bulunması durumunda bu metodun programcı tarafından düzenlenmesi ve eklenen hizmet sınıflarının da parametre olarak iletilmesi gerekmektedir.

Hizmet Sınıfı

Hizmet sınıfları ServiceBase sınıfını miras alan ve hizmet kod gerçekleştiriminin içerisinde bulunduğu sınıflardır. Hizmet sınıfı birebir hizmet kavramına karşılık gelmektedir.

Aşağıda ServiceBase sınıfı üye bilgi ve açıklamaları gösterilmiştir.

| Üye Adı | Açıklama |
| --- | --- |
| AutoLog | Hizmeti durdurma, duraklatma ve başlatma olaylarının otomatik olarak Windows olay günlüğüne yazılıp yazılmayacağını belirtir. |
| CanPauseAndContinue | Hizmetin duraklatılabileceğini belirten değerdir. |
| CanStop | Hizmetin durdurulabileceğini belirten değerdir. |
| EventLog | Olay günlük nesnesini döndüren özelliktir. |
| ExitCode | Hizmetin çıkış kodunu belirtir. Çıkış koduna sıfırdan farklı bir değer atanarak işletim sistemine hizmetin başarısız sonlandığı belirtilebilir. |
| ServiceName | Hizmet adını belirtir. |
| OnContinue | Duraklatılmış hizmetin SCM tarafından tekrar başlatılması durumunda otomatik olarak çağrılan metottur. |
| OnPause | Çalışan hizmetin SCM tarafından tekrar duraklatılması durumunda otomatik olarak çağrılan metottur. |
| OnShutdown | Sistemin kapatılması durumunda otomatik olarak çağrılan metottur. |
| OnStart | Hizmetin SCM tarafından başlatılması durumunda otomatik olarak çağrılan metottur. |
| OnStop | Hizmetin durdurulmasıyla otomatik olarak çağrılan metottur. |
| RequestAdditionalTime | SCM tarafından iletilen komutun hizmet tarafından işlenme süresinin aşılması durumunda çağrılarak ek zaman isteğini belirten metottur. |
| Run | Parametre olarak iletilen hizmet veya hizmetleri çalışabilmeleri için yükler. |
| Stop | Çalışan hizmeti durdurur. |

Tablo 134, System.ServiceProcess.ServiceBase Sınıfı Üye Bilgileri

Hizmet sınıfı özellikleri yapıcı metot içerisinde veya görsel olarak VS.NET tarafından düzenlenebilir.

Ekran Çıktısı 83, VS.NET ile Hizmet Özelliklerinin Düzenlenmesi

Hizmet adı ve hizmet sınıf adının birbirleri ile aynı olma zorunluluğu bulunmamaktadır.

Kullanım Bilgileri

Aşağıdaki kod parçasında örnek hizmet sınıfı ve açıklamalar gösterilmiştir.

C#

**public** **partial** **class** OrnekHizmet : ServiceBase {

**protected** **override** **void** OnStart(**string**[] args) {

// Hizmetin SCM tarafından başlatılması ile çalıştırılacak kodlar.  
 }

**protected** **override** **void** OnStop() {

// Hizmetin SCM tarafından durdurululması ile çalıştırılacak kodlar.

}

**protected** **override** **void** OnPause() {

// Hizmetin SCM tarafından duraklatılması ile çalıştırılacak kodlar.

}

**protected** **override** **void** OnContinue() {

// Duraklatılmış hizmetin SCM tarafından başlatılması ile çalıştırılacak   
 // kodlar.

}

}

VB.NET

**Public** **Class** OrnekHizmet

**Protected** **Overrides** **Sub** OnStart(**ByVal** args() **As** **String**)

**'** Hizmetin SCM tarafından başlatılması ile çalıştırılacak kodlar.  
 **End** **Sub**

**Protected** **Overrides** **Sub** OnStop()

**'** Hizmetin SCM tarafından durdurululması ile çalıştırılacak kodlar.

**End** **Sub**

**Protected** **Overrides** **Sub** OnPause()

**'** Hizmetin SCM tarafından duraklatılması ile çalıştırılacak kodlar.

**End** **Sub**

**Protected** **Overrides** **Sub** OnContinue()

**'** Duraklatılmış hizmetin SCM tarafından başlatılması ile çalıştırılacak   
 ' kodlar.

**End** **Sub**

**End** **Class**

Kod 347, Örnek Hizmet Sınıfı

Hizmet tarafından yapılacak işlemlerin hizmet sınıfı yerine farklı bir sınıf ve iş parçacığı tarafından yapılması önerilmektedir.

Hizmet tarafından yapılması istenen işlemler için aşağıdaki örnekte gösterilen sınıf şablon olarak kullanılabilir.

C#

**class** OrnekHizmetYoneticisi {

**private** Thread isParcacigi;

**private** **bool** duraklatildiMi = **false**;

**internal** OrnekHizmetYoneticisi() {

isParcacigi = **new** Thread(Basla);

}

**internal** **void** Basla() {

**while** (**true**) {

**if** (duraklatildiMi) {

Thread.Sleep(500);

**continue**;

}

// Bu bölümde işlemler yapılabilir.

}

}

**internal** **void** Baslat() {

isParcacigi.Start();

}

**internal** **void** Durdur() {

isParcacigi.Abort();

}

**internal** **void** Duraklat() {

duraklatildiMi = **true**;

}

**internal** **void** DevamEt() {

duraklatildiMi = **false**;

}

}

VB.NET

**Class** OrnekHizmetYoneticisi

**Private** isParcacigi **As** Thread

**Private** duraklatildiMi **As** **Boolean** = **False**

**Friend** **Sub** **New**()

isParcacigi = **New** Thread(**AddressOf** Basla)

**End** **Sub**

**Friend** **Sub** Basla()

**While** **True**

**If** duraklatildiMi **Then**

Thread.Sleep(500)

**Continue** **While**

**End** **If**

' Bu bölümde işlemler yapılabilir.

**End** **While**

**End** **Sub**

**Friend** **Sub** Baslat()

isParcacigi.Start()

**End** **Sub**

**Friend** **Sub** Durdur()

isParcacigi.Abort()

**End** **Sub**

**Friend** **Sub** Duraklat()

duraklatildiMi = **True**

**End** **Sub**

**Friend** **Sub** DevamEt()

duraklatildiMi = **False**

**End** **Sub**

**End** **Class**

Kod 348, Örnek Hizmet Sınıf Kullanımı

Kod parçasında istisna yönetimi gösterilmemiştir. İstisna yönetimi ve olay günlüğü kullanımı hizmet uygulamalarında büyük önem taşımaktadır. Oluşan istisnaların ve bilgilendirici mesajların mutlaka olay günlüğüne yazılması önerilmektedir.

OrnekHizmetYoneticisi nesnesi aşağıdaki şekilde hizmet içerisinden oluşturulabilir ve kullanılabilir.

C#

**public** **partial** **class** OrnekHizmet : ServiceBase {

**private** OrnekHizmetYoneticisi yonetici;

**public** OrnekHizmet() {

InitializeComponent();

yonetici = **new** OrnekHizmetYoneticisi();

}

**protected** **override** **void** OnStart(**string**[] args) {

yonetici.Basla();

}

**protected** **override** **void** OnStop() {

yonetici.Durdur();

yonetici = **null**;

}

**protected** **override** **void** OnPause() {

yonetici.Duraklat();

}

**protected** **override** **void** OnContinue() {

yonetici.DevamEt();

}

}

VB.NET

**Public** **Class** OrnekHizmet

**Private** yonetici **As** OrnekHizmetYoneticisi

**Public** **Sub** **New**()

InitializeComponent()

yonetici = **New** OrnekHizmetYoneticisi()

**End** **Sub**

**Protected** **Overrides** **Sub** OnStart(**ByVal** args() **As** **String**)

yonetici.Basla()

**End** **Sub**

**Protected** **Overrides** **Sub** OnStop()

yonetici.Durdur()

yonetici = **Nothing**

**End** **Sub**

**Protected** **Overrides** **Sub** OnPause()

yonetici.Duraklat()

**End** **Sub**

**Protected** **Overrides** **Sub** OnContinue()

yonetici.DevamEt()

**End** **Sub**

**End** **Class**

Kod 349, OrnekHizmetYoneticisi Sınıfının Hizmet İçerisinden Kullanımı

Hizmet Uygulamalarının Yüklenmesi

Hizmet uygulamalarını SCM hizmet veritabanına eklemek için hizmet uygulaması ve hizmet yükleyicisi (ing. Service Process Installer, Service Installer) sınıfı kullanılmaktadır.

Hizmetin kolaylıkla yüklenebilmesini sağlamak amacıyla kullanılan bu sınıfları oluşturmanın en pratik yolu VS.NET hizmet özelliklerini gösteren pencerede Add Installer komutunun verilmesidir. Komutun verilmesi ile birlikte VS.NET tarafından hizmetin yüklenmesi için gereken sınıf otomatik olarak oluşturulmaktadır.

Ekran Çıktısı 84, Hizmet Yükleyici Sınıfının VS.NET ile Oluşturulması

İşlemin tamamlanması ile birlikte uygulama derlenerek installutil komutu ile hizmet yüklenebilir.

Installutil uygulama.exe

Hata Ayıklama

Hizmet uygulamalarında hata ayıklama (ing. Debug) işleminin yapılabilmesi için hizmetin öncelikle sisteme yüklenmiş ve çalışır olması gerekmektedir. Çalışan hizmet, Debug | Attach to Process … komutu verilerek seçilmeli ve hata ayıklama işlemi yapılmalıdır.

Kıyamete yakın Müslümanlar içinde en az bulunacak şey; helal para ile kendisine güvenilecek arkadaştır.

Hz. Muhammed

Bölüm

23

Veritabanı Programlama ve ADO.NET

G

ünümüzde geliştirilen uygulama yazılımlarının büyük bölümü veri oluşturma ve oluşturulan verinin kullanımı modeline sahiptir. Veritabanı uygulamaları olarak nitelenebilecek bu yazılımlar, verinin kullanıcıdan alınması, doğrulanması, işlenmesi ve ilişkisel veritabanı yönetim sistemleri kullanılarak yönetilmesini içermektedir.

Kitabın bu bölümünde veritabanı uygulamalarının ADO.NET kullanılarak geliştirilmesi konusunda bilgiler sunulacaktır.

Konulara geçmeden önce okuyucunun SQL ve ilişkisel veritabanları konusunda temel bilgi düzeyine sahip olması önerilmektedir.

Yazılım Tasarım Önerileri

ADO.NET ve veritabanı uygulamaları konusuna geçilmeden önce, programcılar tarafından her projede kullanımı önerilen bazı yöntem ve yazılım katmanları hakkında temel düzeyde bilgiler sunulacaktır.

Mantıksal Katmanlar

Mantıksal katmanlar (ing. Logical Layers), yazılım projelerinde farklı işlev ve gerçekleştirimleri birbirlerinden ayırmak, belirli bir disiplin çerçevesinde yazılım geliştirme ve test işlemlerinin yapılabilmesini sağlamaktadırlar.

Üç temel katman, kullanıcı ara yüz katmanı (ing. User Interface), iş katmanı (ing. Business Layer) ve veri katmanı (ing. DataAccess Layer) olarak özetlenebilir.

Temel katmanların tamamı veya bir bölümü tarafından kullanılabilen güvenlik katmanı (ing. Security Layer), konfigürasyon katmanı (ing. Configuration Layer), istisna yönetim katmanı (ing. Exception Management Layer) veya projeye özgü diğer katmanlar projelerde bulunabilir.

Şekil 48, Mantıksal Yazılım Katmanları

Yazılım katmanlarının farklı Assembly dosyaları içerisinde bulunması önerilmekle birlikte, aynı Assembly içerisindeki mantıksal katmanların ayrılmasında ad alanları kullanılabilir.

İş varlıkları (ing. Business Entities), yazılım projesinde kullanılan veriler ve verilerin ilişkisel veritabanı yapısına yansımasını ifade etmektedir. Örneğin bir veritabanı tablosunu simgeleyen sınıf iş varlığına örnek olarak verilebilir.

Mantıksal katmanların uygulama yazılımlarında kullanımında dikkat edilmesi gereken konulardan birisi de katmanlar arasındaki ilişkidir. Genel prensip, kullanıcı ara yüz katmanından iş katmanı içerisinde bulunan iş nesnelerinin (ing. Business Objects) kullanımıdır. İş nesneleri, gerektiği durumda veri katmanı içerisinde bulunan veri erişim nesnelerini (ing. Data Access Objects) kullanmalıdır. Ara yüz katmanından direkt veri erişim katmanı içerisindeki nesnelerin kullanımı önerilmemektedir.

Ara Yüz Katmanı

Ara yüz katmanı, kullanıcı tarafından bilgi girişi yapılan ve bilginin kullanıcıya iletildiği katmandır. Bu katmanın temel işlevi girdi ve çıktının kullanıcı ile paylaşımı olarak özetlenebilir.

Örneğin bir Windows uygulamasında kullanıcı formları, kontrolleri ve raporlar kullanıcı ara yüz katmanının parçalarıdır. Benzer şekilde bir web uygulamasında, web sayfaları temel kullanıcı ara yüz katmanını oluşturmaktadır.

Ara yüz katmanında herhangi bir iş mantığı gerçekleştirimi önerilmemektedir. Ara yüz katmanı, kullanıcı tarafından girilen bilgilerin iş katmanına, iş katmanı tarafından üretilen çıktıların kullanıcıya iletilmesi amacıyla kullanılmalıdır. Temel veri doğrulama (ing. Data Validation) işlemlerinin bu katmanda yapılmasında herhangi bir sakınca bulunmamaktadır.

İş Katmanı

İş katmanı, yazılımın amacı olan iş gereksinimleri ve mantığının bulunduğu katmandır. Bu katman içerisinde iş mantığına ek olarak veri doğrulama, yetki kontrolü, işlem grubu (ing. Transaction) ve istisna yönetimi gibi işlemlerin de yapılması mümkündür.

Bu katman içerisinde bulunan iş nesnelerinin olabildiğince bağımsız geliştirilmesi önerilmektedir. Bağımsızlık kavramı ile, uygulama türüne göre bu katmanın değiştirilmesinin gerekmemesi ifade edilmektedir. Örneğin aynı projenin Windows, web ve mobil uygulamaları içerdiği değerlendirildiğinde, farklı ara yüz katmanları aynı iş katmanını kullanabilmelidir.

Veri Erişim Katmanı

Veri erişim katmanı, veritabanı bağlantısı ve veri erişimlerinin yapıldığı katmandır. Bu katmanın kullanım amaçlarından birisi de iş katmanını kullanılan veritabanından olabildiğince bağımsızlaştırmaktır.

Örnek Mimari

Yukarıda temel kuralları ve işlevleri ile aktarılan mantıksal katmanların pratikte kullanımını göstermek amacıyla hazırlanan aşağıdaki örnek VS.NET çözüm yapısını inceleyiniz.

Ekran Çıktısı 85, Mantıksal Katmanlar ve VS.NET Projeleri

Örnek VS.NET çözümü, iş ve veri erişim katmanları ile kullanıcıların web, Windows ve PDA cihazları ile uygulamayı kullanabilmesini sağlayan ara yüz katmanlarından oluşmaktadır.

Aşağıda katmanların birbirleri ile ilişkileri gösterilmiştir.

Şekil 49, Mantıksal Katman İlişkileri

Veritabanı Erişim Yöntemleri ve ADO.NET

Veriye erişimin standartlaştırılması amacıyla farklı çözümler bulunmaktadır. Bu çözümler ile hedeflenen, verinin kaynağı ile uygulama yazılımı arasına yeni bir katman ekleyerek veritabanı işlemlerinin standart yöntemler ile yapılabilmesini sağlamaktır.

Veritabanı Erişim Yöntemleri

Microsoft tarafından sunulan ODBC (Open Database Connectivity) ve Borland tarafından sunulan BDE (Borland Database Engine) bu katmanın öncü örneklerindendir. Gereksinimlerin artması ve yöntemlerin geliştirilmesi ile Microsoft tarafından OleDb (Object Linking and Embedding Database) kullanımıma sunulmuştur.

Şekil 50, Veritabanı Uygulaması ve Veritabanı

Veritabanı kavramından sadece ilişkisel veritabanı yönetim sistemleri anlaşılmamalıdır. Veritabanı ile verinin saklandığı herhangi bir ortam ifade edilmektedir.

Her veritabanı üreticisi, Microsoft tarafından belirlenen OleDb standartlarına uyarak kendi veritabanlarına erişim için OleDb sağlayıcı (ing. OLEDB Provider) yazılımları geliştirmiştir. Böylece uygulama yazılımları OleDb standartlarında veriye erişirken, üretici tarafından sağlanan OleDb sağlayıcısı ile istenilen veritabanına erişim mümkün olmaktadır.

Şekil 51, OleDb Sağlayıcıları

ADO (ActiveX Database Objects), Microsoft ActiveX teknolojisi kullanılarak gerek ODBC sürücüleri gerek OLE-DB sağlayıcıları ile haberleşebilen kütüphanedir. Microsoft tarafından geliştirilen ADO içerisinde bulunan nesneler yardımıyla Delphi veya ASP (Active Server Pages) kullanılarak OleDb sağlayıcılarına erişim mümkün olmaktadır.

Şekil 52, ADO ve Veritabanı Erişimi

ADO.NET

ADO.NET, yukarıda anlatılan ADO mimarisinin geliştirilerek .NET Framework sınıfları ile gerçekleştirimidir. Bir başka anlatımla ADO.NET, .NET uygulamalarında veri erişim katmanı veya kütüphanesi olarak tanımlanabilir. ADO mimarisinden farklı olarak ADO.NET mimarisi içerisindeki sınıf ve nesnelerin yönetilen ortam (ing. Managed) içerisinde çalıştığına dikkat edilmelidir.

ADO.NET mimarisi aşağıda gösterilmiştir.

Şekil 53, ADO.NET Mimarisi

ADO.NET direkt veritabanı erişimini (ing. Native Database Access) desteklemektedir. Direkt veritabanı erişimi, veritabanı ile ADO.NET arasında herhangi bir katman bulunmadığını ifade etmektedir. Farklı üreticiler tarafından geliştirilen .NET veri sağlayıcıları direkt veritabanı erişimi için kullanılmaktadır.

.NET Veri Sağlayıcıları

ADO.NET kütüphanesi ile veritabanı arasında bulunan katman .NET veri sağlayıcısı (ing. .NET Data Provider) olarak adlandırılmaktadır.

.NET Framework içerisinde aşağıda listelenen veri sağlayıcıları bulunmaktadır.

| Veri Sağlayıcısı | Açıklama |
| --- | --- |
| SQL Server Veri Sağlayıcısı | SQL Server veritabanına direkt erişim amacıyla kullanılan sağlayıcıdır ve System.Data.SqlClient ad alanı içerisinde bulunmaktadır. |
| OleDb Veri Sağlayıcısı | OleDb üzerinden veri erişimi için kullanılan sağlayıcıdır ve System.Data.OleDb ad alanı içerisinde bulunmaktadır. |
| ODBC Veri Sağlayıcısı | ODBC üzerinden veri erişimi için kullanılan sağlayıcıdır ve System.Data.Odbc ad alanı içerisinde bulunmaktadır. |
| Oracle Veri Sağlayıcısı | Oracle veritabanına direkt erişim amacıyla kullanılan sağlayıcıdır ve System.Data.OracleClient ad alanı içerisinde bulunmaktadır. |

Tablo 135, .NET Framework Veri Sağlayıcıları ve Açıklamaları

Oracle veri sağlayıcısı System.Data.OracleClient.dll içerisinde bulunmaktadır.

.NET Framework içerisinde bulunan sağlayıcılara ek olarak, farklı üreticiler tarafından kendi veritabanlarına .NET uygulamalarından direkt erişim amacıyla kullanılabilecek veri sağlayıcıları geliştirilmiştir.

Aşağıda bazı veritabanları ve direkt veritabanı erişimi için kullanılan .NET veri sağlayıcıları listelenmiştir.

| Veritabanı | Sağlayıcı |
| --- | --- |
| Advantage | Advantage .NET Data Provider (Ads.NET) |
| MySQL | MySQLDirect .NET Data Provider |
| Oracle | Oracle .NET Data Provider (Odp.NET) |
| PostgreSQL | PostgreSQLDirect .NET Data Provider |
| VistaDb | VistaDb .NET Data Provider |

Tablo 136, Farklı Veritabanları ve .NET Veri Sağlayıcıları

.NET veri sağlayıcısı ile OleDb veri sağlayıcısı birbirlerine karıştırılmamalıdır. .NET veri sağlayıcısı, ADO.NET mimarisinde, OleDb veri sağlayıcısı OleDb tarafından belirlenen standartlara uyularak geliştirilmiş sağlayıcıdır. ADO.NET ile OleDb sağlayıcısına erişim için .NET OleDb Veri sağlayıcısı kullanılmalıdır.

.NET veri sağlayıcıları \Data Providers klasöründe bulunmaktadır.

Veri Sağlayıcısı Seçimi

Veriye .NET uygulamalarından ADO.NET standartlarında erişim için kullanılacak .NET veri sağlayıcısının seçimi büyük önem taşımaktadır. Veri sağlayıcısının, veri işlemleri performansına ve geliştirme süresine önemli etkileri olabilmektedir.

Zorunlu olmadıkça .NET ODBC veri sağlayıcısının kullanımı önerilmemektedir. Uygulamanın farklı veritabanlarını destekler şekilde geliştirilme gereksiniminin bulunduğu durumlarda .NET OleDb veri sağlayıcısı seçilmelidir. .NET OleDb veri sağlayıcısı, kullanılan veritabanı yönetim sisteminin direkt erişim sağlayan .NET veri sağlayıcısının bulunmadığı durumlarda da tercih edilmelidir.

Çoğunlukla, en yüksek performans ve kullanım kolaylığı direkt erişim sağlayan .NET veri sağlayıcıları ile elde edilebilmektedir. Veri sağlayıcısı seçiminde aşağıda listelenen öneriler değerlendirilmelidir.

1. Genel prensip olarak, direkt veritabanı erişimi sağlayan veri sağlayıcıları tercih edilmelidir.
2. Direkt veri erişiminin istenmediği durumlarda .NET OleDb veri sağlayıcısı tercih edilmelidir.
3. Aynı veritabanına direkt erişim sağlayan farklı veri sağlayıcılarının bulunması durumunda veritabanı üreticisi önerileri değerlendirilmelidir.
4. Eğer mümkünse, seçim yapılmadan önce genel fikir sağlayabilecek denemeler farklı sağlayıcılar ile yapılmalıdır.

ADO.NET Mimarisi

ADO.NET iki temel bileşen üzerine kurulmuştur.

* .NET Veri Sağlayıcıları
* Veri Kümeleri (ing. Datasets)

.NET veri sağlayıcıları veriye erişim için, veri kümeleri verinin işlenmesi amacıyla kullanılmaktadır.

System.Data ve altında bulunan ad alanları içerisinde ADO.NET mimarisi sınıf ve tipleri bulunmaktadır.

ADO.NET Sınıfları

ADO.NET mimarisinde kullanılan temel sınıflar ve açıklamaları aşağıda gösterilmiştir.

| Sınıf | Açıklama |
| --- | --- |
| DbConnection | Veri tabanı bağlantısını yöneten temel bağlantı sınıfıdır. Sınıf, IDbConnection arabirim gerçekleştirimini yapmaktadır. |
| DbCommand | Veritabanı komutunu simgeleyen temel komut sınıfıdır. Veritabanı komutu SQL cümlesi veya depolanmış prosedür (ing. Stored Procedure) veya tablo adı olabilir. Sınıf, IDbCommand arabirim gerçekleştirimini yapmaktadır. |
| DbDataReader | SQL sorgusu sonucunda dönen bilgileri tek yönlü okumayı sağlayan temel veri okuyucu sınıfıdır. Sınıf, IDbDataReader arabirim gerçekleştirimini yapmaktadır. |
| DbDataAdapter | Veri kümeleri ile veritabanı arasındaki bağlantıyı sağlayan temel veri adaptörü sınıftır. Sınıf, IDbDataAdapter arabirim gerçekleştirimini yapmaktadır. |
| DbParameter | SQL cümlelerinde parametreyi simgeleyen temel parametre sınıfıdır. Sınıf, IDbParameter ve IDbDataParameter arabirim gerçekleştirimlerini yapmaktadır. |
| DbTransaction | İşlem grubunu (ing. Transaction) simgeleyen temel sınıftır. Sınıf, IDbTransaction arabirim gerçekleştirimini yapmaktadır. |

Tablo 137, ADO.NET Mimarisi Sınıfları ve Açıklamaları

Yukarıda listelenen soyut sınıflar farklı .NET veri sağlayıcıları tarafından miras alınmaktadır. Örneğin System.Data.SqlClient ad alanı içerisinde bulunan ve SQL Server veritabanına direkt veritabanı bağlantısı yapmak için kullanılan SqlConnection, DbConnection sınıfını miras almaktadır. Benzer şekilde SqlCommand sınıfı DbCommand sınıfını miras almaktadır.

Böylece, tüm .NET veri sağlayıcı sınıfları için temel gerçekleştirimin yapıldığı bir yapı sunulmaktadır.

Bağlantısız Model

Bağlantısız model (Disconnected Model), veritabanı bağlantısının sadece gerektiği durumda ve kısa süreli olarak yapılması ve kapatılmasını ifade etmektedir.

Bağlantısız model ile elde edilen kazanım sistem performansının artması ve genişleyebilirliğin (ing. Scalibility) üst düzeyde sağlanmasıdır. Veritabanı yönetim sistemleri aktif her veritabanı bağlantısı için belirli sistem kaynakları ayırmaktadır. Sistem kullanıcılarının yüksek sayıda olması ve her kullanıcı için veritabanı bağlantısının açık tutulması sistem kaynakların gereksiz yere kullanımını beraberinde getirmektedir.

Örneğin istemci – sunucu (ing. Client – Server) mimari, uygulamanın açılışında veritabanı bağlantısının yapılıp uygulama kapanana dek bağlantının açık tutulmasını temel almaktadır. Yapısı gereği iki katmandan oluşan bu mimari ile hedeflenen, az sayıda kullanıcıya kapalı bir ağ içerisinde hizmet vermek iken çok katmanlı web ve zengin istemci mimarisinde hedeflenen, çok daha yüksek sayıdaki kullanıcıya en optimum şekilde hizmet vermektir.

Bağlantı Havuzu

Bağlantısız model, herhangi bir veritabanı işleminde veritabanı bağlantısının açılmasını, işlenmek istenen verinin bilgisayar hafızasına alınmasını ve hemen ardından veritabanı bağlantısının kapatılmasını esas almaktadır. Bilgisayar hafızasında veri işlendikten sonra, örneğin veride değişiklik yapıldığında, bağlantının tekrar açılması, SQL cümlelerinin veritabanına iletilmesi ve bağlantının kapatılması bu modelin genel prensibini oluşturmaktadır.

Veritabanı bağlantılarının sadece gerektiğinde yapılıyor olması, bağlantı yapma ve bağlantıyı kapatma işleminin sık yapılması gereğini beraberinde getirmektedir. Bu durumun herhangi bir performans olumsuzluğuna yol açmaması için bağlantı havuzları (ing. Connection Pool) kullanılmaktadır.

Bağlantı havuzunu destekleyen .NET veri sağlayıcıları, uygulama yazılımından veritabanı bağlantısı yapılmak istendiğinde, havuzda hazır bulunan bir veritabanı bağlantısını kullanmaktadırlar. Böylece veritabanı bağlantı yapılması süresi elimine edilebilmektedir.

Bağlantı havuzu kullanımında dikkat edilmesi gereken en önemli konu, veritabanı bağlantı parametrelerinin yapılan tüm bağlantılar için aynı olması gerekliliğidir. Veritabanı bağlantı parametreleri kullanıcı adı, şifreleri ve çeşitli değerleri içerebilmektedir. Bu değerlerin yapılan her bağlantı için farklı olması bağlantı havuzunun sağladığı avantajların ortadan kalkmasına sebep olmaktadır.

Çoğunlukla, istemci – sunucu mimaride uygulamalar geliştirilirken kullanıcının belirttiği kullanıcı adı ve şifresi (kimlik doğrulama bilgileri) ile veritabanına bağlantı yapılmaktadır. Bağlantısız modelde kullanıcı kimlik doğrulama bilgileri ve uygulamanın veritabanına bağlantıda kullandığı kullanıcı adı ve şifresi birbirinden ayrılmalıdır. Önerilen yöntem, uygulamanın yapacağı tüm veritabanı bağlantılarının aynı kullanıcı adı ve şifresi ile (aynı bağlantı parametreleri ile) yapılmasıdır. İletilen kullanıcı kimlik doğrulama bilgilerinin LDAP (Lightweight Directory Access Protocol) veya veritabanında bu amaçla tutulan bir tablo aracılığı ile kontrol edilmesi önerilmektedir.

Bağlantı havuzunun desteklenmemesi veya yukarıda aktarılan yöntemlerin uygulanmamasından dolayı kullanılamaması önemli performans problemlerine yol açabilecektir.

Veri Kümeleri

Veri kümeleri bağlantısız modelin en önemli parçalarıdır. Veri kümeleri, veritabanından çekilen verinin bilgisayar hafızasında kolaylıkla işlenebilmesini, arama ve veri üzerinde değişiklik yapılabilmesini sağlamaktadır.

Veri kümeleri, çok katmanlı mimari içerisinde iş varlıklarına karşılık gelmektedir.

Uygulama yazılımlarında iş varlıkları olarak veri kümelerinin kullanılma zorunluluğu bulunmamaktadır. Çoğu uygulama geliştirme aracı ve platformunda, kullanıcı kontrolleri normal ve veri bazlı olarak kategorize edilmiş olmasına rağmen (örneğin Delphi için TEdit sadece bilgi almak için, TDbEdit veritabanı verisi ile çalışmak için kullanılmaktadır. Veritabanı verisinin mutlaka TDataset sınıfını miras alan bir nesne tarafından sağlanma zorunluluğu bulunmaktadır), .NET veri bağlama (.NET Data Binding) mimarisi, kullanıcı kontrollerinin istenilen her nesne verisine bağlanabilmesini sağlamaktadır. Veri kümeleri dışında olan ve programcı tarafından geliştirilen iş varlık sınıfları Custom Business Entity olarak adlandırılmaktadır.

\Yardımcı Araçlar\MyGeneration klasörü içerisinde veritabanı tablolarından iş varlıklarının oluşturulabilmesi için otomatik kod üretimi yapabilen MyGeneration aracı bulunmaktadır.

System.Data.DataSet sınıfı temel veri kümesi sınıfıdır. Bir veri kümesi içerisinde farklı veritabanı tablo ve veri ilişki nesneleri bulunabilmektedir.

İnsanların birbirini tanıması için en iyi zaman, ayrılmalarına en yakın zamandır.

Fyodor Mikailoviç Dostoyevski

Bölüm

24

Veritabanı Kullanım Yöntemleri

Ö

nceki bölümde ADO.NET ve mimarisi konularında verilen bilgilerin ardından, ADO.NET kullanılarak temel veritabanı işlemlerinin yapılması ve çeşitli örnekler bu bölümün konusunu oluşturmaktadır.

Örnek kod parçalarında SQL Server veri sağlayıcısı kullanılmıştır. Buna rağmen, diğer .NET veri sağlayıcılarında da işlemlerin yapılması, kullanımı ve çıktılarının aynı olacağı unutulmamalıdır. Örneğin SqlConnection yerine OracleConnection, SqlCommand yerine OracleCommand veya SqlDataAdapter yerine OracleDataAdapter sınıfları kullanıldığında aynı işlemler kolaylıkla yapılabilecektir.

Veritabanı Bağlantısı Yönetimi

System.Data.Common.DbConnection sınıfı veritabanı bağlantısının yönetimi amacıyla kullanılan sınıftır.

Aşağıda sınıf üye bilgi ve açıklamaları listelenmiştir.

| Üye Adı | Açıklama |
| --- | --- |
| ConnectionString | Veritabanı bağlantı dizesini belirtmektedir. |
| ConnectionTimeout | Bağlantı zaman aşımı süresini belirtmektedir. |
| Database | Veritabanı adını belirtmektedir. |
| DataSource | Veri kaynak adını belirtmektedir. |
| State | Bağlantı durumunu belirtmektedir. |
| ServerVersion | Bağlantı yapılan sunucu sürümünü belirtmektedir. |
| BeginTransaction | Yeni bir işlem grubu başlatır. |
| ChangeDatabase | Veritabanı bağlantısı açıkken bağlanılan veritabanı değiştirmek amacıyla kullanılmalıdır. |
| Close | Veritabanı bağlantısını kapatır. |
| CreateCommand | Veritabanına SQL cümleleri iletmek amacıyla kullanılabilecek komut nesnesi döndürür. |
| Open | Veritabanı bağlantısını açar. |

Tablo 138, System.Data.Common.DbConnection Sınıfı Üye Bilgileri

Soyut bir sınıf olan DbConnection, farklı .NET veri sağlayıcı sınıfları tarafından miras alınmaktadır. Miras alan sınıf, kullanılan veritabanı yönetim sistemi özelliklerine göre yukarıda listelenen çeşitli özellik değerlerini belirlemektedir.

Veritabanı Bağlantı Dizesi

Veritabanı bağlantı dizesi (ing. Database Connection String), bağlantı parametrelerini içeren karakter dizi değerdir. Bağlantı dizesi içerisinde veritabanı sunucu adresi, bağlanılmak istenen veritabanı, kullanıcı adı şifresi, bağlantı havuzu değerleri gibi kullanılan .NET veri sağlayıcısı tarafından desteklenen farklı bilgiler bulunabilmektedir.

Bağlantı dizesinin kitapta daha önce anlatılan yöntemlerle korunması önerilmektedir.  
  
Farklı veritabanları için kullanılacak veritabanı bağlantı dizesi değerleri için kullandığınız veritabanı yönetim sistemi dokümantasyonunu inceleyiniz.

Aşağıda örnek veritabanı bağlantı dizeleri gösterilmiştir. Bağlantı dizesi içerisindeki değerler ; karakteri ile ayrılmaktadır.

Data Source=KLT;User ID=kullanici;Password=s;Unicode=True  
  
Data Source=KLT;Initial Catalog=Vagon;Persist Security Info=True;User ID=kullanici;Password=s

VS.NET Sunucu Gezgini

Farklı .NET veri sağlayıcıları kullanılarak tasarım anında veritabanı bağlantısı yapılması ve veri yapısının yönetimi VS.NET sunucu gezgini penceresi yardımıyla mümkündür.

Ekran Çıktısı 86, VS.NET Veri Bağlantıları Penceresi

Yeni bir bağlantı yapmak için Data Connections elemanı üzerinde sağ tıklayarak Add Connection … komutu verilmelidir. Gelen pencere yardımıyla .NET veri sağlayıcısının ve sağlayıcıya özgü bağlantı parametrelerinin belirtilmesi mümkündür.

Ekran Çıktısı 87, VS.NET ile Yeni Veritabanı Bağlantısı Eklenmesi

Change komutu verilerek sistemde yüklü .NET veri sağlayıcıları listelenebilmekte ve seçilen sağlayıcı kullanılabilmektedir.

Ekran Çıktısı 88, NET ile Sistemde Yüklü .NET Veri Sağlayıcılarını Listeleme

Advanced … komutu, sağlayıcı tarafından desteklenen parametreleri görsel olarak düzenlemek amacıyla kullanılabilir. Ad – değer ikililerinden oluşan parametreler, veritabanı bağlantı dizesinde ; ile birbirinden ayrılmaktadır.

Ekran Çıktısı 89, VS.NET ile Veritabanı Bağlantı Parametrelerini Düzenleme

Sunucu gezginine veri katmanı bağlantısı eklendikten sonra, eklenen bağlantı seçilerek bağlantı özellikleri VS.NET özellikler penceresi ile görüntülenebilir.

Ekran Çıktısı 90, .NET Özellikler Penceresi ve Veritabanı Bağlantısı

Özellikler penceresinde Connection String değeri veritabanı bağlantı dizesini belirtmektedir. Bu yöntemle görsel olarak bağlantı dizesi değerini elde etmek mümkündür.

Kullanılan .NET veri sağlayıcısının desteklemesi durumunda tüm veritabanı işlemleri (tablo oluşturma, veri görüntüleme / düzenleme…) VS.NET içerisinden yapılabilmektedir.

Bağlantının Açılması ve Kapatılması

Aşağıda veritabanı bağlantısının SqlConnection nesnesi ile açılması ve kapatılmasının örneklendiği kod parçası bulunmaktadır. İşlem diğer .NET veri sağlayıcıları için de benzerdir.

C#

**using** (SqlConnection conn = **new** SqlConnection("Bağlantı Dizesi")) {

// Bağlantıyı aç.

conn.Open();

// Bağlantıyı kapat.

conn.Close();

}

VB.NET

**Using** conn **As** SqlConnection = **New** SqlConnection("Bağlantı Dizesi")

' Bağlantıyı aç.

conn.Open()

' Bağlantıyı kapat.

conn.Close()

**End** **Using**

Kod 350, Veritabanı Bağlantısının Açılması ve Kapatılması

DbConnection sınıfı IDisposable arabirim gerçekleştirimini yaptığı için using bloğu kullanımı önerilmektedir.

Bağlantı havuzu kullanılması durumunda bağlantının açılması işlemi çok kısa sürelerde yapılabilmektedir. Bağlantının açık kalma süresinin olabildiğince kısa tutulması, havuzdan alınan bağlantının en kısa sürede tekrar havuza gönderilmesini sağlayacaktır.

Veri Erişim İşlemleri

ADO.NET veri erişimi ve işlemleri farklı yöntemlerle yapılabilmektedir. Kullanılabilecek yöntemler hakkında bu bölümde bilgiler verilecektir.

Tasarım Önerileri

Veritabanı tasarımı yapılırken ve uygulamanın veri erişim altyapısı kurulurken değerlendirilebilecek bazı öneriler aşağıda sunulmuştur.

1. Veritabanı yönetim sisteminin desteklediği durumda tablo isimlerinin büyük küçük harflerden oluşması ve olabildiğince anlaşılabilir olması önerilmektedir.
2. Özel durumlar dışında, her tabloya Id isimli ve değeri otomatik artan tamsayı tipinde olan bir alan eklenmesi ve bu alanın ana anahtar (Primary Key) olarak belirtilmesi önerilmektedir.
3. Uygulama kodu içerisine SQL cümlelerinin gömülmesi önerilmemektedir. Veritabanı yönetim sisteminin desteklediği durumda tüm veri işlemlerinin depolanmış prosedürler ile yapılması önerilmektedir.
4. Depolanmış prosedür kullanımında en sık yapılan hatalardan birisi iş mantığını depolanmış prosedürler ile gerçekleştirmektir. Çok katmanlı mimari kuralları gereği, iş mantığının tamamı uygulama yazılımı iş katmanı tarafından gerçekleştirilmeli, veri katmanı sadece verinin elde edilmesi ve kaydedilmesi amacıyla kullanılmalıdır.
5. Uygulama yazılımından veritabanı depolanmış prosedürünün çağrımında parametre kullanımı benimsenmelidir. Değer içeren SQL cümlesi karakter dizilerinin (SQL String) uygulama yazılımı tarafından oluşturulması ve veritabanı yönetim sistemine iletilmesi önerilmemektedir.

Veri Modeli

Konunun anlatımında aşağıda gösterilen veri yapısı kullanılacaktır. Veri modeli SQL Server veritabanı yönetim sistemi ile oluşturulmuştur.

Tablo Bilgileri

Veri modelinde kullanılan tablolar ve tablolar arasındaki ilişkiler aşağıda gösterilmiştir.

Şekil 54, Örnek Veri Modeli

Depolanmış Prosedür Bilgileri

Veriye erişim ve veri düzenleme işlemlerinde kullanılmak üzere hazırlanan bazı depolanmış prosedürler ve kullanım amaçları aşağıda gösterilmiştir.

| Prosedür Adı | Açıklama |
| --- | --- |
| EkleKisi | Kişi tablosuna yeni bir kayıt eklemek amacıyla kullanılmaktadır. |
| GuncelleKisi | Kişi tablosundaki bir kaydı güncellemek amacıyla kullanılmaktadır. |
| SilKisi | Kişi tablosundan kayıt silmek amacıyla kullanılmaktadır. |
| OkuKisi | Kayıt numarası (Id) ile kişi bilgisine erişmek amacıyla kullanılmaktadır. |
| AraKisi | Ad veya soyadı bilgisine göre kişi kayıtlarına erişmek amacıyla kullanılmaktadır. |
| AraKisiSayfali | Ad veya soyadı bilgisine göre kişi kayıtlarına sayfalayarak erişmek amacıyla kullanılmaktadır. |
| GetirKisiSayisi | Ad veya soyadı bilgisine göre arama yapmak ve uyan kayıt sayısını elde etmek amacıyla kullanılmaktadır. |
| OkuTumSehir | Tüm şehir kayıtlarını elde etmek amacıyla kullanılmaktadır. |

Tablo 139, Depolanmış Prosedür İsimleri ve Açıklamaları

Örnek olması açısından bazı depolanmış prosedür içerikleri aşağıda gösterilmiştir.

**Create** **Procedure** [EkleKisi]

( @Id int **output**, @TcKimlikNo bigint, @Ad nvarchar(25), @Soyad nvarchar(25),  
 @DogumTarih datetime

) **AS INSERT** **INTO** [Kisi] ([TcKimlikNo], [Ad], [Soyad], [DogumTarih])  
 VALUES(@TcKimlikNo, @Ad, @Soyad, @DogumTarih)

**SELECT** @Id = SCOPE\_IDENTITY();

Kod 351, EkleKisi Depolanmış Prosedür İçeriği

**Create** **Procedure** [GuncelleKisi]

( @Id int, @TcKimlikNo bigint, @Ad nvarchar(25), @Soyad nvarchar(25),  
 @DogumTarih datetime

) **AS UPDATE** [Kisi] **SET** [TcKimlikNo] = @TcKimlikNo,

[Ad] = @Ad,

[Soyad] = @Soyad,

[DogumTarih] = @DogumTarih

**WHERE** [Id] = @Id

Kod 352, GuncelleKisi Depolanmış Prosedür İçeriği

**Create** **Procedure** [SilKisi]

( @Id int )

**AS DELETE** **FROM** [Kisi] **WHERE** [Id] = @Id

Kod 353, SilKisi Depolanmış Prosedür İçeriği

**Create** **Procedure** [OkuKisi]

( @Id int)

**AS SELECT** \* **FROM** [Kisi] **WHERE** [Id] = @Id

Kod 354, OkuKisi Depolanmış Prosedür İçeriği

**Create** **Procedure** [AraKisi]

( @Ad nvarchar(25), @Soyad nvarchar(25), @ KayitSirasi nvarchar(15) )

**AS**

**BEGIN**

**SELECT** [Id], [TcKimlikNo], [Ad], [Soyad], [DogumTarih]   
 **FROM** [Kisi]

**WHERE** (@Ad **IS** **NULL** **OR** [Ad] **LIKE** @Ad +'%') **AND**

(@Soyad **IS** **NULL** **OR** [Soyad] **LIKE** @Soyad +'%')

**ORDER** **BY** **CASE** @KayitSirasi **WHEN** 'Ad' **THEN** Ad **END**,

**CASE** @ KayitSirasi **WHEN** 'Soyad' **THEN** Soyad **END**

**END**

Kod 355, AraKisi Depolanmış Prosedür İçeriği

**Create** **Procedure** [GetirKisiSayisi]

( @Ad nvarchar(50), @Soyad nvarchar(50) )

**AS**

**BEGIN**

**SELECT** **count**(\*) **FROM** [Kisi]

**WHERE** (@Ad **IS** **NULL** **OR** [Ad] **LIKE** @Ad +'%') **AND**

(@Soyad **IS** **NULL** **OR** [Soyad] **LIKE** @Soyad +'%')

**END**

Kod 356, GetirKisiSayisi Depolanmış Prosedür İçeriği

Gösterilen veri yapısı ve depolanmış prosedür kodları   
\Bölüm 24\VeriModeli klasöründe bulunmaktadır.

Veritabanı Komut Sınıfı Kullanımı

Veritabanı komut sınıfı, veritabanı yönetim sistemine SQL cümlelerinin iletilmesi amacıyla kullanılmaktadır. İletilen SQL cümleleri kayıt ekleme, silme veya güncelleme gibi işlemleri yapan veri döndürmeyen tipte olabileceği gibi, tablo veya görünümden veri döndüren tipte de olabilmektedir.

Veritabanı komut sınıfının sık kullanılan üye bilgileri aşağıda gösterilmiştir.

| Üye Adı | Açıklama |
| --- | --- |
| CommandText | Veritabanı komutunu belirtir. Komut, SQL cümlesi, depolanmış prosedür adı veya tablo adı olabilmektedir. |
| CommandTimeout | Komutun çalıştırılma zaman aşımı süresini belirtir. |
| CommandType | Komut tipini belirtir. |
| Connection | Komutun kullandığı veritabanı bağlantı nesnesini belirtir. |
| Parameters | Komut parametrelerini belirtir. |
| Transaction | Komutun içerisinde çalıştığı işlem grubu nesnesini belirtir. |
| Cancel | Komutun çalışmasını iptal eder. |
| CreateParameter | Komut parametresi oluşturur ve oluşturulan parametre nesnesini döndürür. |
| ExecuteNonQuery | Sonuç döndürmeyen (ekleme, silme, vb.) komutları çalıştırır. |
| ExecuteReader | Sonuç döndüren komutu çalıştırır ve veri okuyucu nesnesi döndürür. |
| ExecuteScalar | Sonuç döndüren komutu çalıştırır ve ilk satır / sütun değerini döndürür. |
| Prepare | Komutu çalıştırılmak üzere veritabanı üzerinde hazırlar. |

Tablo 140, Veritabanı Komut Sınıfı Üye Bilgileri

Örnek kod parçalarında kullanılan conn nesnesi önceki konu başlığında oluşturulan veritabanı bağlantı nesnesini ifade etmektedir. Veritabanı bağlantısının Open metodu ile açılmış olduğu varsayılmıştır.

Komut Nesneleri Oluşturma

Aşağıdaki kod parçasında komut nesnelerinin oluşturulması gösterilmiştir.

C#

SqlCommand cmd = **new** SqlCommand("[Komut]", conn);

VB.NET

**Dim** cmd **As** SqlCommand = **New** SqlCommand("[Komut]", conn)

Kod 357, Veritabanı Komut Nesnesi Oluşturma

Komut nesneleri parametresiz yapıcı metot çağrılarak da oluşturulabilmektedir. Bu durumda CommandText ve Connection özelliklerine sırasıyla komut ve bağlantı nesnesi değerleri atanmalıdır.

Komut nesnesi oluşturmanın bir başka yöntemi de bağlantı nesnesinin CreateCommand metodunun çağrılmasıdır.

Komut Tipinin Belirtilmesi

System.Data.CommandType sayılabilir tipi komut tipini belirtmek amacıyla kullanılmaktadır.

| C# | VB.NET |
| --- | --- |
| **public** **enum** CommandType {  // Depolanmış prosedür  StoredProcedure,  // Tablo adı  TableDirect,  // SQL cümlesi  Text  } | **Public** **Enum** CommandType  **'** Depolanmış prosedür  StoredProcedure  ' Tablo adı  TableDirect  ' SQL cümlesi  [Text]  **End** **Enum** |

Kod 358, System.Data.CommandType Sayılabilir Tip Tanımı

DbCommand.CommandType özelliği, DbCommand.CommandText özelliği ile ifade edilen komutun tipini belirtmektedir.

Veritabanı yönetim sistemleri sadece SQL komutlarını işleyebilmektedir. Komut tipinin CommandType.Text değeri dışında olması durumunda, komut nesnesi tarafından SQL cümlesi hazırlanmaktadır. Örneğin komut değeri sadece tablo adından oluşuyorsa ve komut tipi CommandType.TableDirect ise select \* from [TabloAdı] SQL cümlesi veritabanına iletilmektedir. Benzer şekilde komut adı sadece depolanmış prosedür adı ise, örneğin, Execute [ProsedürAdı] veritabanına iletilmektedir.

Aşağıdaki örnekte farklı komutların belirtilmesi gösterilmiştir.

C#

SqlCommand cmd = **new** SqlCommand("Kisi", conn);

cmd.CommandType = CommandType.TableDirect;

cmd.CommandText = "SilKisi";

cmd.CommandType = CommandType.StoredProcedure;

cmd.CommandText = "select \* from Kisi";

cmd.CommandType = CommandType.Text;

VB.NET

**Dim** cmd **As** SqlCommand = **New** SqlCommand("Kisi", conn)

cmd.CommandType = CommandType.TableDirect

cmd.CommandText = "SilKisi"

cmd.CommandType = CommandType.StoredProcedure

cmd.CommandText = "**select** \* from Kisi"

cmd.CommandType = CommandType.Text

Kod 359, Veritabanı Nesnesi Komut Değerleri ve Tipinin Belirtilmesi

Komut Parametreleri Kullanımı

Komut parametreleri, veritabanına iletilmek istenen SQL cümlesine istenilen değerlerin dinamik olarak eklenebilmesini sağlamaktır. Örneğin Kisi tablosundan bir kayıt silmek için kayıt numarası bilgisi veya arama yaparken kullanılan kriterler komut parametrelerine örnek olarak verilebilir.

Komut parametreleri kullanılmadan, SQL cümlesi içerisine değerlerin eklenmesi mümkündür. Aşağıda bu yöntemin kullanıldığı örnek kod parçası gösterilmiştir.

C#

**string** sql = "select \* from kisi where ad = '";

Console.WriteLine("Arama yapmak istediğiniz adı giriniz:");

**string** ad = Console.ReadLine();

sql = sql + ad + "'";

SqlCommand cmd = **new** SqlCommand(sql, conn);

VB.NET

**Dim** sql **As** **String** = "**select** \* from kisi where ad = '"

Console.WriteLine("Arama yapmak istediğiniz adı giriniz:")

**Dim** ad **As** **String** = Console.ReadLine

sql = sql + ad + "'"

**Dim** cmd **As** SqlCommand = **New** SqlCommand(sql, conn)

Kod 360, Komut Parametreleri Kullanılmadan SQL Cümlesi Oluşturma

Örnek kod parçasında, karakter dizi değerlerinin birleştirilmesi ile oluşturulan SQL cümlesi veritabanına iletilmektedir. Sık yapılan hatalardan olan ve sistem performans ve güvenliğini büyük ölçüde düşürebilen bu yöntemin kullanımı hiçbir zaman tercih edilmemelidir. Yöntem aşağıdaki dezavantajları beraberinde getirmektedir.

1. SQL Injection atakları adı verilen ve kötü niyetli kişinin girdi içerisine istediği komutu yerleştirerek veri kayıplarına yol açabilmesini ifade eden ataklara sistem açık olmaktadır. Yukarıdaki örnekte hiçbir doğrulamadan geçirilmeyen ad isimli yerel değişken değerinin direkt veritabanına iletildiğine dikkat ediniz.
2. Her arama için farklı SQL cümleleri oluşturulması, dolayısıyla oluşturulan SQL cümlelerinin veritabanı yönetim sistemi tarafından tekrar işlenmesinin sistem performansına negatif etkileri olabilmektedir.

Listelenen olumsuzlukları elimine etmek amacıyla komut parametreleri kullanılmalıdır.

ADO.NET içerisinde komut parametresi System.Data.Common.DbParameter soyut sınıfı ile ifade edilmektedir. Komut parametresi üç temel özelliğe sahiptir.

* Parametre Adı, SQL cümlesi içerisindeki parametre adı.
* Parametre Tipi, parametrenin tipi.
* Parametre Yönü, Parametrenin girdi, çıktı veya her iki yönde olduğu.

Parametre yönü (ing. Parameter Direction), uygulama ve veritabanı arasında parametre değerinin iletim ve dönüş yönünü belirtmektedir. Girdi parametre (ing. Input Parameter), değeri veritabanına iletilen parametreleri belirtmek için kullanılmaktadır. Komut parametreleri kullanımı ile veritabanı tarafından döndürülen değerlerin elde edilmesi mümkündür. Bu parametrelere çıktı parametreler (ing. Output Parameters) adı verilmektedir. Parametre değerinin veritabanına iletimi ve veritabanı tarafından değişen değerin tekrar uygulamaya iletimi için girdi-çıktı parametreleri (ing. Input – Output Parameters) kullanılmaktadır. DbParameter.Value özelliğine atanan değer parametrenin değeri olmaktadır.

SQL cümlesi içerisinde birden çok parametre bulunabilmektedir. Komut nesnesi, komut parametrelerini DbParameterCollection isimli koleksiyon sınıfı ile yönetmektedir.

Aşağıdaki kod parçasında SQL Server veritabanı için komut parametresi kullanımı gösterilmiştir.

C#

**string** sql = "select \* from kisi where ad = @ad";

Console.WriteLine("Arama yapmak istediğiniz adı giriniz:");

**string** ad = Console.ReadLine();

SqlCommand cmd = **new** SqlCommand(sql, conn);

// Parametre adı, tipi ve boyutu ile parametre nesnesi oluştur.

SqlParameter p = **new** SqlParameter("@ad", SqlDbType.NVarChar, 25);

p.Value = ad;

cmd.Parameters.Add(p);

VB.NET

**Dim** sql **As** **String** = "select \* from kisi where ad = @ad"

Console.WriteLine("Arama yapmak istediğiniz adı giriniz:")

**Dim** ad **As** **String** = Console.ReadLine()

**Dim** cmd **As** SqlCommand = **New** SqlCommand(sql, conn)

' Parametre adı, tipi ve boyutu ile parametre nesnesi oluştur.

**Dim** p **As** SqlParameter = **New** SqlParameter("@ad", SqlDbType.NVarChar, 25)

p.Value = ad

cmd.Parameters.Add(p)

Kod 361, SQL Server Komut Parametresi Kullanım Örneği

SQL Server veritabanı için parametreler @ karakteri ile belirtilmektedir. Farklı veritabanlarında SQL cümlesi içerisindeki parametrenin ayrımı için farklı karakterler kullanılmalıdır. Aşağıdaki örnekte Oracle veritabanı için benzer işlev gösterilmiştir.

C#

**string** sql = "select \* from KISI where AD = :ad";

Console.WriteLine("Arama yapmak istediğiniz adı giriniz:");

**string** ad = Console.ReadLine();

OracleCommand cmd = **new** OracleCommand(sql, conn);

OracleParameter p = **new** OracleParameter("ad", OracleType.NVarChar, 25);

p.Value = ad;

cmd.Parameters.Add(p);

VB.NET

**Dim** sql **As** **String** = "select \* from KISI where AD = :ad"

Console.WriteLine("Arama yapmak istediğiniz adı giriniz:")

**Dim** ad **As** **String** = Console.ReadLine()

**Dim** cmd **As** OracleCommand = **New** OracleCommand(sql, conn)

**Dim** p **As** OracleParameter = **New** OracleParameter("ad", OracleType.NVarChar, 25)

p.Value = ad

cmd.Parameters.Add(p)

Kod 362, Oracle Komut Parametresi Kullanımı

SQL cümlesi içerisinde bulunan tüm parametrelerin parametre nesneleri olarak komuta eklenmesi gerekmektedir.

Sonuç Döndürmeyen SQL Cümleleri Kullanımı

Sonuç döndürmeyen SQL cümlelerinin veritabanına iletimi için DbCommand.ExecuteNonQuery metodu kullanılmalıdır. Metot, yapılan işlem sonucunda etkilenen kayıt sayısını döndürmektedir.

C#

SqlCommand cmd = **new** SqlCommand("EkleKisi", conn);

// Komutun depolanmış prosedürü ifade ettiğini belirt.

cmd.CommandType = CommandType.StoredProcedure;

// Parametreleri oluştur.

SqlParameter pId = **new** SqlParameter("@Id", SqlDbType.Int);

pId.Direction = ParameterDirection.Output;

SqlParameter pTcKimlikNo = **new** SqlParameter("@TcKimlikNo", SqlDbType.BigInt);

SqlParameter pAd = **new** SqlParameter("@Ad", SqlDbType.NVarChar, 25);

SqlParameter pSoyad = **new** SqlParameter("@Soyad", SqlDbType.NVarChar, 25);

SqlParameter pDogumTarih=**new** SqlParameter("@DogumTarih", SqlDbType.DateTime);

// Parametre değerlerini belirle.

pTcKimlikNo.Value = 59716221182;

pAd.Value = "Ahmet";

pSoyad.Value = "Yılmaz";

pDogumTarih.Value = **new** DateTime(1955, 11, 11);

// Parametreleri komuta ekle.

cmd.Parameters.Add(pId);

cmd.Parameters.Add(pTcKimlikNo);

cmd.Parameters.Add(pAd);

cmd.Parameters.Add(pSoyad);

cmd.Parameters.Add(pDogumTarih);

// SQL cümlesini veritabanında çalıştır.

**int** kayitSayisi = cmd.ExecuteNonQuery();

Console.WriteLine("Etkilenen Kayıt Sayısı: {0}", kayitSayisi);

Console.WriteLine("Üretilen Kayıt Numarası: {0}", pId.Value);

VB.NET

**Dim** cmd **As** SqlCommand = **New** SqlCommand("EkleKisi", conn)

' Komutun depolanmış prosedürü ifade ettiğini belirt.

cmd.CommandType = CommandType.StoredProcedure

' Parametreleri oluştur.

**Dim** pId **As** SqlParameter = **New** SqlParameter("@Id", SqlDbType.Int)

pId.Direction = ParameterDirection.Output

**Dim** pTcKimlikNo **As** SqlParameter = **New** SqlParameter("@TcKimlikNo", \_  
 SqlDbType.BigInt)

**Dim** pAd **As** SqlParameter = **New** SqlParameter("@Ad", SqlDbType.NVarChar, 25)

**Dim** pSoyad **As** SqlParameter=**New** SqlParameter("@Soyad", SqlDbType.NVarChar, 25)

**Dim** pDogumTarih **As** SqlParameter = **New** SqlParameter("@DogumTarih", \_  
 SqlDbType.DateTime)

' Parametre değerlerini belirle.

pTcKimlikNo.Value = 59716221182

pAd.Value = "Ahmet"

pSoyad.Value = "Yılmaz"

pDogumTarih.Value = **New** DateTime(1955, 11, 11)

' Parametreleri komuta ekle.

cmd.Parameters.Add(pId)

cmd.Parameters.Add(pTcKimlikNo)

cmd.Parameters.Add(pAd)

cmd.Parameters.Add(pSoyad)

cmd.Parameters.Add(pDogumTarih)

' SQL cümlesini veritabanında çalıştır.

**Dim** kayitSayisi **As** **Integer** = cmd.ExecuteNonQuery()

Console.WriteLine("Etkilenen Kayıt Sayısı: {0}", kayitSayisi)

Console.WriteLine("Üretilen Kayıt Numarası: {0}", pId.Value)

Kod 363, DbCommand.ExecuteNonQuery Kullanımı

Kod çıktısı aşağıda gösterilmiştir.

Etkilenen Kayıt Sayısı: 1

Üretilen Kayıt Numarası: 1

Komutun başarıyla çalıştırılması ile Kisi isimli tabloya bir kayıt eklenmektedir. VS.NET sunucu gezgini penceresinde tablo üzerinde sağ tıklayarak verilen Show Table Data komutuyla tablo verisi görüntülenebilir.

Ekran Çıktısı 91, Eklenen Kaydın Görüntülenmesi

Sonuç Döndüren SQL Sorguları ve Veri Okuyucular

Veri okuyucular, sonuç döndüren SQL sorgularının (ing. SQL Query) veritabanında çalıştırılması sonucu uygulamaya iletilen kayıtların okunabilmesini sağlamak amacıyla kullanılmaktadır.

Sonuç döndüren SQL cümlelerinin veritabanında çalıştırılması ve sonuçların uygulamaya iletimi farklı işlemlerdir. Komut çalıştırılması (ing. Command Execution), SQL cümlesinin veritabanı yönetim sistemi tarafından işlenerek oluşturulan imleç bilgisinin (ing. Cursor) uygulamaya iletimini, kayıtların uygulama tarafından alınması ise kayıt çekme (ing. Record Fetch) olarak adlandırılmaktadır.

Temel veri okuyucu sınıfı olan DbDataReader üye bilgi ve açıklamaları aşağıda gösterilmiştir.

| Üye Adı | Açıklama |
| --- | --- |
| FieldCount | SQL cümlesi sonucunda dönen sonuç kümesindeki alan sayısını (sütun sayısını) döndürür. |
| HasRows | Sonuç kümesinin kayıt içermesi durumunda True, kayıt içermemesi durumunda False değerini döndürür. |
| IsClosed | Veri okuyucunun kapalı olması durumunda True, açık olması durumunda False değerini döndürür. |
| Item | Aktif kayıt bilgisine sütun numarası veya adı ile erişmek amacıyla kullanılmalıdır. |
| Close | Veri okuyucuyu kapatır. |
| GetFieldType | Parametre olarak iletilen sütun numarasında bulunan alan tipini System.Type nesnesi olarak döndürür. |
| GetXXX | Aktif kayıt bilgisine sütun numarası ile erişmek için kullanılmaktadır. Örneğin GetInt32 alan değerini tamsayıya dönüştürerek döndürmektedir. |
| GetName | Parametre olarak iletilen sütun numarasına göre sütun adını döndürür. |
| GetValue | Aktif kayıt bilgisine parametre olarak iletilen sütun numarası ile erişmek için kullanılmalıdır. |
| GetValues | Aktif kayıt değerlerini parametre olarak iletilen nesne dizisi içerisine almak için kullanılmalıdır. |
| IsDbNull | Parametre olarak iletilen sütun numarasında herhangi bir değer olup olmadığını döndürür. |
| Read | Bir sonraki kayda konumlanmayı sağlar. Değerinin False dönmesi durumunda son kayda konumlama yapıldığı anlaşılmalıdır. |

Tablo 141, System.DataCommon.DbDataReader Sınıfı Üye Bilgileri

Veri okuyucu nesnesi DbCommand.ExecuteReader metodu tarafından döndürülmektedir. Üye bilgilerinden de görülebileceği gibi veri okuyucusu sadece ileri doğru okuma yapabilmektedir.

C#

SqlCommand cmd = **new** SqlCommand("AraKisi", conn);

cmd.CommandType = CommandType.StoredProcedure;

// Komut parametrelerini oluştur.

SqlParameter pAd = **new** SqlParameter("@Ad", SqlDbType.NVarChar, 25);

SqlParameter pSoyad = **new** SqlParameter("@Soyad", SqlDbType.NVarChar, 25);

SqlParameter pKayitSirasi = **new** SqlParameter("@KayitSirasi",   
 SqlDbType.NVarChar, 15);

// Parametre değerlerini belirle.

pAd.Value = DBNull.Value;

pSoyad.Value = "Y";

pKayitSirasi.Value = "Ad";

// Parametre nesnenerini ekle.

cmd.Parameters.Add(pAd);

cmd.Parameters.Add(pSoyad);

cmd.Parameters.Add(pKayitSirasi);

// Veri okuyucu nesnesini oluştur.

SqlDataReader reader = cmd.ExecuteReader();

VB.NET

**Dim** cmd **As** SqlCommand = **New** SqlCommand("AraKisi", conn)

cmd.CommandType = CommandType.StoredProcedure

' Komut parametrelerini oluştur.

**Dim** pAd **As** SqlParameter = **New** SqlParameter("@Ad", SqlDbType.NVarChar, 25)

**Dim** pSoyad **As** SqlParameter=**New** SqlParameter("@Soyad", SqlDbType.NVarChar, 25)

**Dim** pKayitSirasi **As** SqlParameter = **New** SqlParameter("@KayitSirasi", \_  
 SqlDbType.NVarChar, 15)

' Parametre değerlerini belirle.

pAd.Value = DBNull.Value

pSoyad.Value = "Y"

pKayitSirasi.Value = "Ad"

' Parametre nesnenerini ekle.

cmd.Parameters.Add(pAd)

cmd.Parameters.Add(pSoyad)

cmd.Parameters.Add(pKayitSirasi)

' Veri okuyucu nesnesini oluştur.

**Dim** reader **As** SqlDataReader = cmd.ExecuteReader()

Kod 364, Veri Okuyucu Oluşturma Örneği

Yukarıdaki örnek kod parçasında AraKisi isimli depolanmış prosedürün çağrılması ve dönen sonucu alabilmek için veri okuyucu nesnesinin oluşturulması gösterilmiştir.

Veri okuyucunun kullanımı ardından mutlaka DbDataReader.Close metodu çağrılarak kapatılması gerekmektedir.

C#

**static** **void** Goster(DbDataReader okuyucu) {

Console.WriteLine("Dönen sütun sayısı: {0}", okuyucu.FieldCount);

// Alan isimlerini yazdır.

**for** (**int** i = 0; i < okuyucu.FieldCount; i++) {

Console.Write("{0}\t", okuyucu.GetName(i));

}

Console.WriteLine();

// Kayıt dönmemesi durumunda uyarı mesajını yazdır.

**if** (!okuyucu.HasRows) {

Console.WriteLine("Kayıt bulunmamaktadır");

} **else** {

// DbDataReader.Read metodu kullanarak sonuç kümesi sonuna kadar oku.

**while** (okuyucu.Read()) {

// Her bir alan değerini yazdır.

**for** (**int** i = 0; i < okuyucu.FieldCount; i++)

Console.Write("{0}\t", okuyucu[i]);

Console.WriteLine();

}

}

// İşlem tamamlandıktan sonra okuyucuyu kapat.

okuyucu.Close();

}

VB.NET

**Shared** **Sub** Goster(**ByVal** okuyucu **As** DbDataReader)

Console.WriteLine("Dönen sütun sayısı: {0}", okuyucu.FieldCount)

' Alan isimlerini yazdır.

**For** i **As** **Integer** = 0 **To** okuyucu.FieldCount - 1

Console.Write("{0}" & vbTab, okuyucu.GetName(i))

**Next**

Console.WriteLine()

' Kayıt dönmemesi durumunda uyarı mesajını yazdır.

**If** **Not** okuyucu.HasRows **Then**

Console.WriteLine("Kayıt bulunmamaktadır")

**Else**

' DbDataReader.Read metodu kullanarak sonuç kümesi sonuna kadar oku.

**While** okuyucu.Read

' Her bir alan değerini yazdır.

**For** i **As** **Integer** = 0 **To** okuyucu.FieldCount - 1

Console.Write("{0}" & vbTab, okuyucu(i))

**Next**

Console.WriteLine()

**End** **While**

**End** **If**

' İşlem tamamlandıktan sonra okuyucuyu kapat.

okuyucu.Close()

**End** **Sub**

Kod 365, Veri Okuyucu Kullanım Örneği

Örnekte, soyadı Y ile başlayan kayıtları ad sırasında listelemek için AraKisi isimli depolanmış prosedür çağrılmıştır. Bir kaydın bu kritere uyduğu Kisi tablosu içeriği ve kod çıktısı aşağıda gösterilmiştir.

Dönen sütun sayısı: 5

Id TcKimlikNo Ad Soyad DogumTarih

1 59716221122 Ahmet Yılmaz 11.11.1955 00:00:00

Veri okuyucu ExecuteReader metodu System.Data.CommandBehavior sayılabilir tipini parametre olarak alabilmektedir. Komut davranışının belirlenebileceği bu değer .NET veri sağlayıcısı tarafından çeşitli optimizasyonlar yapılması amacıyla kullanılabilmektedir.

| C# | VB.NET |
| --- | --- |
| **public** **enum** CommandBehavior {  CloseConnection,  Default,  KeyInfo,  SchemaOnly,  SequentialAccess,  SingleResult,  SingleRow  } | **Public** **Enum** CommandBehavior  CloseConnection  [Default]  KeyInfo  SchemaOnly  SequentialAccess  SingleResult  SingleRow  **End** **Enum** |

Kod 366, System.Data.CommandBehavior Tip Tanımı

Aşağıdaki tabloda kullanılabilecek değer ve açıklamaları gösterilmiştir.

| Üye Adı | Açıklama |
| --- | --- |
| CloseConnection | Veri okuyucunun kapatılması ile veritabanı bağlantısının da otomatik olarak kapatılmasını sağlar. |
| Default | Ön tanımlı değerdir. |
| KeyInfo | Sorgunun, bir tabloya ait sütun ve ana anahtar bilgilerini döndüreceğini belirtir. |
| SchemaOnly | Sorgunun veritabanı yapısı bilgilerini döndüreceğini belirtir. |
| SequentialAccess | İçerisinde BLOB (Binary Large Object) bulunan sorguların optimizasyonu amacıyla kullanılmalıdır. |
| SingleResult | İletilen sorgu sonucunda tek kayıt dönmesinin beklendiğini ifade eder. |
| SingleRow | İletilen sorgu sonucunda tek kayıt dönmesinin beklendiğini ve işlem sonucunda veritabanında değişiklik olabileceğini belirtir. |

Tablo 142, System.Data.CommandBehavior Üye Bilgileri

Tahmin edilebileceği durumlarda komut davranış değerinin belirtilmesi önerilmektedir. Böylece kullanılan .NET veri sağlayıcısı performans artırıcı bazı optimizasyonlar yapabilmektedir. Örneğin OkuKisi isimli depolanmış prosedür ile sadece bir kayıt dönebilmektedir. Bu prosedürün uygulamadan çağrımında CommandBehavior.SingleResult değerinin iletimi sağlayıcı tarafından gerekli optimizasyonun yapılmasını sağlayabilecektir.

C#

SqlCommand cmd = **new** SqlCommand("OkuKisi", conn);

cmd.CommandType = CommandType.StoredProcedure;

SqlParameter pId = **new** SqlParameter("@Id", SqlDbType.Int);

pId.Value = 1;

cmd.Parameters.Add(pId);

SqlDataReader reader = cmd.ExecuteReader(CommandBehavior.SingleResult |   
 CommandBehavior.CloseConnection);

reader.Read();

Console.WriteLine("Id: {0}", reader.GetInt32(0));

Console.WriteLine("TcKimlik No: {0}", reader.GetInt64(1));

Console.WriteLine("Ad: {0}", reader.GetString(2));

Console.WriteLine("Soyad: {0}", reader.GetString(3));

Console.WriteLine("Doğum Tarihi: {0}", reader.GetDateTime(4));

// Okuyucunun kapatılması ile veritabanı bağlantısı da otomatik olarak  
// kapatılacaktır.

reader.Close();

VB.NET

**Dim** cmd **As** SqlCommand = **New** SqlCommand("OkuKisi", conn)

cmd.CommandType = CommandType.StoredProcedure

**Dim** pId **As** SqlParameter = **New** SqlParameter("@Id", SqlDbType.Int)

pId.Value = 1

cmd.Parameters.Add(pId)

**Dim** reader **As** SqlDataReader = cmd.ExecuteReader(CommandBehavior.SingleResult\_  
 **Or** CommandBehavior.CloseConnection)

reader.Read()

Console.WriteLine("Id: {0}", reader.GetInt32(0))

Console.WriteLine("TcKimlik No: {0}", reader.GetInt64(1))

Console.WriteLine("Ad: {0}", reader.GetString(2))

Console.WriteLine("Soyad: {0}", reader.GetString(3))

Console.WriteLine("Doğum Tarihi: {0}", reader.GetDateTime(4))

' Okuyucunun kapatılması ile veritabanı bağlantısı da otomatik olarak  
' kapatılacaktır.

reader.Close()

Kod 367, System.Data.CommandBehavior Kullanım Örneği

Tekil Değerler

DbDataCommand.ExecuteScalar metodu veritabanından tek değer elde etmek için kullanılmaktadır. Örneğin GetirKisiSayisi isimli depolanmış prosedür, iletilen parametre değerlerine uyan kaç kayıt olduğunu döndürmektedir. Bu prosedürün çağrımında ExecuteScalar metodunun kullanımı üst düzey optimizasyonun veri sağlayıcısı tarafından yapılabilmesini sağlayacaktır.

C#

SqlCommand cmd = **new** SqlCommand("GetirKisiSayisi", conn);

cmd.CommandType = CommandType.StoredProcedure;

SqlParameter pAd = **new** SqlParameter("@Ad", SqlDbType.NVarChar, 25);

SqlParameter pSoyad = **new** SqlParameter("@Soyad", SqlDbType.NVarChar, 25);

pAd.Value = "A";

pSoyad.Value = DBNull.Value;

cmd.Parameters.Add(pAd);

cmd.Parameters.Add(pSoyad);

**int** sayi = (**int**)cmd.ExecuteScalar();

VB.NET

**Dim** cmd **As** SqlCommand = **New** SqlCommand("GetirKisiSayisi", conn)

cmd.CommandType = CommandType.StoredProcedure

**Dim** pAd **As** SqlParameter = **New** SqlParameter("@Ad", SqlDbType.NVarChar, 25)

**Dim** pSoyad **As** SqlParameter=**New** SqlParameter("@Soyad", SqlDbType.NVarChar, 25)

pAd.Value = "A"

pSoyad.Value = DBNull.Value

cmd.Parameters.Add(pAd)

cmd.Parameters.Add(pSoyad)

**Dim** sayi **As** **Integer** = **CType**(cmd.ExecuteScalar(), **Integer**)

Kod 368, DbCommand.ExecuteScalar Kullanım Örneği

ExecuteScalar metodu System.Object tipinde bir nesne döndürmektedir. Böylece bu metot ile istenilen tipte değerin elde edilmesi mümkün olmaktadır.

Veri Adaptör Sınıfı Kullanımı

ADO.NET mimarisinde veri adaptörlerinin temel kullanım amacı, veri kümelerine otomatik olarak kayıtların yüklenebilmesi, veri kümeleri üzeride yapılan değişikliklerin otomatik olarak veritabanına iletilebilmesidir. Veri adaptörleri bu özellikleri ile ADO.NET tarafından sağlanan veri kaydetme katmanı (ing. Data Persistency Layer) olarak nitelenebilir.

Uygulama yazılımlarında kullanılabilecek farklı veri kaydetme katmanları bulunmaktadır. Hibernate.Net bu katmanların en popüler ve açık kaynak kodlu olan kütüphanelerindendir. Hibernate.Net hakkında daha fazla bilgi www.hibernate.org sitesinde bulunabilir.

Bilindiği gibi veritabanı uygulamalarında iş varlıkları olarak ADO.NET tarafından sunulan veri kümelerini kullanma zorunluluğu bulunmamaktadır. Veri kümelerinin kullanılmaması durumunda veri adaptörleri kullanımı da gerekmemektedir.

Temel veri adaptör sınıfı olan DbDataAdapter üye bilgileri aşağıda gösterilmiştir.

| Üye Adı | Açıklama |
| --- | --- |
| DeleteCommand | Veri kümelerinden silinen kayıtların veritabanına iletimi için kullanılan komut nesnesini belirtir. |
| InsertCommand | Veri kümelerine eklenen kayıtların veritabanına iletimi için kullanılan komut nesnesini belirtir. |
| SelectCommand | Kayıtları seçmek için kullanılan komut nesnesini belirtir. |
| UpdateCommand | Veri kümelerinde güncellenen kayıtların veritabanına iletimi için kullanılan komut nesnesini belirtir. |
| Fill | Parametre olarak iletilen veri kümesi veya tablosu nesnesini SelectCommand özelliği ile ifade edilen komut nesnesini kullanarak doldurur. |
| Update | Parametre olarak iletilen veri kümesi veya tablosu nesneleri içerisindeki kayıt değişikliklerini veritabanına iletir. |

Tablo 143, System.Data.Common.DbAdapter Sınıfı Üye Bilgileri

Veri Adaptör Nesneleri Oluşturma

Aşağıdaki kod parçasında veri adaptör nesnesi oluşturulması gösterilmiştir.

C#

SqlCommand secKomutu = **new** SqlCommand("select \* from Kisi", conn);

// Kayıtları seçmek için kullanılmak istenen komut nesnesini ilet.

SqlDataAdapter adapter = **new** SqlDataAdapter(secKomutu);

// veya

// Sadece komut değerinin iletilmesi ile kayıtları seçmek için   
// kullanılan komut nesnesi otomatik olarak oluşturulmaktadır.

adapter = **new** SqlDataAdapter("select \* from Kisi", conn);

VB.NET

' Kayıtları seçmek için kullanılmak istenen komut nesnesini ilet.

**Dim** secKomutu **As** SqlCommand = **New** SqlCommand("**select** \* from Kisi", conn)

' veya

' Sadece komut değerinin iletilmesi ile kayıtları seçmek için   
' kullanılan komut nesnesi otomatik olarak oluşturulmaktadır.

**Dim** adapter **As** SqlDataAdapter = **New** SqlDataAdapter(secKomutu)

adapter = **New** SqlDataAdapter("**select** \* from Kisi", conn)

Kod 369, Veri Adaptör Nesneleri Oluşturmak

Adaptör Komutlarını Otomatik Olarak Oluşturmak

Veri kümesi içerisinde silinen, güncellenen ve eklenen kayıtların veritabanına iletiminde sırasıyla DeleteCommand, UpdateCommand ve InsertCommand özellikleri kullanılmaktadır. Bu özelliklere, işlevlerine göre istenilen komut nesneleri atanabilmektedir.

DbCommandBuilder nesnesi, silme, güncelleme ve ekleme komut nesnelerinin otomatik olarak oluşturulması amacıyla kullanılabilmektedir. Otomatik komut üretimi için DbDataAdapter nesnesinin SelectCommand özelliği kullanılmaktadır.

C#

SqlCommand secKomutu = **new** SqlCommand("select \* from Kisi", conn);

SqlDataAdapter adapter = **new** SqlDataAdapter(secKomutu);

SqlCommandBuilder cb = **new** SqlCommandBuilder(adapter);

SqlCommand silKomutu = cb.GetDeleteCommand();

SqlCommand guncelleKomutu = cb.GetUpdateCommand();

SqlCommand ekleKomutu = cb.GetInsertCommand();

Console.WriteLine(silKomutu.CommandText);

Console.WriteLine(guncelleKomutu.CommandText);

Console.WriteLine(ekleKomutu.CommandText);

VB.NET

**Dim** secKomutu **As** SqlCommand = **New** SqlCommand("**select** \* from Kisi", conn)

**Dim** adapter **As** SqlDataAdapter = **New** SqlDataAdapter(secKomutu)

**Dim** cb **As** SqlCommandBuilder = **New** SqlCommandBuilder(adapter)

**Dim** silKomutu **As** SqlCommand = cb.GetDeleteCommand

**Dim** guncelleKomutu **As** SqlCommand = cb.GetUpdateCommand

**Dim** ekleKomutu **As** SqlCommand = cb.GetInsertCommand

Console.WriteLine(silKomutu.CommandText)

Console.WriteLine(guncelleKomutu.CommandText)

Console.WriteLine(ekleKomutu.CommandText)

Kod 370, Adaptör Komutlarının Otomatik Olarak Oluşturulması

Kod çıktısı aşağıda gösterilmiştir.

DELETE FROM [Kisi] WHERE (([Id] = @p1) AND ([TcKimlikNo] = @p2) AND ([Ad] = @p3)

AND ([Soyad] = @p4) AND ([DogumTarih] = @p5))

UPDATE [Kisi] SET [TcKimlikNo] = @p1, [Ad] = @p2, [Soyad] = @p3, [DogumTarih] =

@p4 WHERE (([Id] = @p5) AND ([TcKimlikNo] = @p6) AND ([Ad] = @p7) AND ([Soyad] =

@p8) AND ([DogumTarih] = @p9))

INSERT INTO [Kisi] ([TcKimlikNo], [Ad], [Soyad], [DogumTarih]) VALUES (@p1, @p2,

@p3, @p4)

Veri Kümeleri

ADO.NET tarafından sunulan bağlantısız modelin en önemli parçalarından birisi veri kümeleridir. Veri kümeleri, verinin uygulama tarafından veritabanı yönetim sisteminden alınarak işlenebilmesini, işlenen verinin veri adaptörleri ile tekrar veritabanına iletimini sağlamaktadırlar.

İş varlıkları olarak uygulama yazılımlarında veri kümelerinin tercih edilmesi, yazılım geliştirme süresini büyük ölçüde azaltabilmekte ancak performansta da bazı düşmelere sebep olabilmektedir.

Genel Mimari

Bağlantısız model ile temelde hedeflenen, veritabanı bağlantısı yapılmadan olabildiğince çok işlevin veri kümeleri ile yapılabilmesidir. Bu kapsamda, veritabanında SQL cümleleriyle yapılabilecek kayıt ekleme ve arama gibi temel işlevlere ek olarak veri ilişkileri, indeks ve ana anahtar değerleri, kısıtlamalar gibi bir çok yapının da veri kümeleriyle yapılabilmesi sağlanmaktadır.

Veri kümesi üç temel yapı taşından oluşmaktadır.

| Yapı Taşı | Açıklama |
| --- | --- |
| Veri Tabloları | Bir veri kümesi içerisinde bir veya daha fazla veri tablosu bulunabilmektedir. Her tablo System.Data.DataTable nesnesi ile ifade edilmektedir. |
| İlişkiler | Veri kümesi içerisinde bulunan tabloların birbirleriyle ilişkileri yönetilebilmektedir. System.Data.DataRelation sınıfı ilişkileri simgeleyen sınıftır. |
| Programcı Tanımlı Özellikler | Programcı tarafından veri kümesi ile ilgili değerin saklanmasını sağlayan özelliklerdir. DataSet.ExtendedProperties bu özelliklerin yönetiminde kullanılmaktadır. |

Tablo 144, Veri Kümesi Yapı Taşları

Veri kümeleri içerisinde birden fazla tablo bulunabilmektedir. Bu anlamda veri kümeleri, birbirleriyle ilişkili tablo veya verilerin mantıksal kapsayıcıları olarak değerlendirilebilir. Uygulama yazılımları tasarımında, her veri kümesinin sadece bir tablo içermesinin bir standart olarak benimsenmesi de mümkündür.

Bağlantısız model gerçekleştiriminde kullanılan sınıflar aşağıda gösterilmiştir.

| Sınıf | Açıklama |
| --- | --- |
| Constraint | Temel kısıt sınıfıdır. |
| DataColumn | Veri tablosu içerisindeki alanı simgeleyen sınıftır. |
| DataRow | Veri tablosu içerisindeki satırı (kaydı) simgeleyen sınıftır. |
| DataView | Sıralama, filtreleme ve arama yapabilmek için kullanılan veri görünüm sınıfıdır. |

Tablo 145, Bağlantısız Model Yardımcı Sınıf ve Açıklamaları

Sınıf Üye Bilgileri

Veri kümeleri kullanım detaylarına başlamadan önce temel sınıflar ve üyeleri hakkında bilgiler sunulacaktır.

Veri Kümesi Sınıfı

Temel veri kümesi System.Data.DataSet sınıfı üye bilgileri ve açıklamaları aşağıda gösterilmiştir.

| Üye Adı | Açıklama |
| --- | --- |
| CaseSensitive | Veri kümesinin sahip olduğu tablolarda yapılan karakter dizi karşılaştırma işlemlerinde büyük küçük harf ayrımı yapılıp yapılmayacağını belirtir. |
| DataSetName | Veri kümesini tanımlayan karakter dizi değerdir. |
| EnforceConstaints | Veri düzenleme işlemlerinde kısıtların uygulanıp uygulanmayacağını belirtir. |
| ExtendedProperties | Programcı tanımlı özellikleri belirtir. |
| Relations | Veri tabloları ilişkilerini belirtir. |
| Tables | Veri kümesi içerisinde bulunan veri tablosu nesnelerini belirtir. |
| AcceptChanges | Veri kümesi verilerine yapılan değişikliklerin onaylanmasını sağlar. |
| Clear | Tüm tablo verilerini temizler. |
| Clone | Veri kümesi yapısını kopyalar ve yeni veri kümesi nesnesi döndürür. |
| Copy | Veri kümesi yapısı ile birlikte verileri de kopyalayarak yeni bir veri kümesi nesnesi döndürür. |
| GetChanges | Veri kümesi verilerinde yapılan değişiklikleri döndürür. |
| GetXml | Veri kümesi verisini XML yapısında döndürür. |
| HasChanges | Veri kümesi verisinde değişiklik olup olmadığını döndürür. |
| Load | Parametre olarak iletilen veri okuyucu nesnesini kullanarak veri kümesini doldurur. |
| Merge | Parametre olarak iletilen veri kümesi nesnesi verisi ile mevcut veri kümesi verisini birleştirir. |
| ReadXml | Parametre olarak iletilen XML verisini okur ve veri kümesine ekler. |
| RejectChanges | Veri kümesi verisinde yapılan değişiklikleri iptal eder. |
| Reset | Veri kümesini orijinal durumuna getirir. |
| WriteXml | Veri kümesi verisini XML olarak yazar. |

Tablo 146, System.Data.DataSet Sınıfı Üye Bilgileri

Veri Tablosu Sınıfı

Temel veri tablosu System.Data.DataTable sınıfı üye bilgileri ve açıklamaları aşağıda gösterilmiştir. Veri kümesinin çoğu üyesinin çağrılmasıyla veri kümesi tarafından yapılan işlem, veri kümesi içerisinde bulunan tüm veri tablosu nesnelerinin aynı isimli üyelerinin çağrımıdır.

| Üye Adı | Açıklama |
| --- | --- |
| Columns | Tablo alanlarını belirtir. |
| Constraints | Tablo kısıtlarını belirtir. |
| DataSet | Tablonun içerisinde bulunduğu veri kümesi nesnesini döndürür. |
| DefaultView | Ön tanımlı veri görünüm nesnesini döndürür. |
| DisplayExpression | Kullanıcı ara yüz kontrollerinde gösterilmek istenen tablo adını belirtir. |
| HasErrors | Tablo içerisinde bulunan satırların başarısız bir işlemden dolayı hata içerip içermediğini belirtir. |
| Rows | Tablo satırlarını belirtir. |
| TableName | Tablo adını belirtir. |
| GetErrors | Hatalı olan satırları döndürür. |
| ImportRow | Parametre olarak iletilen satır nesnesini tabloya ekler. |
| NewRow | Tabloya yeni satır ekler. |
| Select | İletilen filtreleme kriterine uyan satırları döndürür. |

Tablo 147, System.Data.DataTable Sınıfı Üye Bilgileri

Sütun Sınıfı

Veri tablosunda bir alanı simgeleyen System.Data.DataColumn sınıfı üye bilgi ve açıklamaları aşağıda gösterilmiştir.

| Üye Adı | Açıklama |
| --- | --- |
| AllowDbNull | Sütunun boş değerlere izin verip vermediğini belirtir. |
| AutoIncrement | Sütunun otomatik artan sayı tipinde olup olmadığını belirtir. |
| Caption | Kullanıcı ara yüz kontrollerinde kullanılan sütun başlık değerini döndürür. |
| ColumnName | Sütun adını belirtir. |
| DataType | Sütun veri tipini belirtir. |
| DefaultValue | Sütun ön tanımlı değerini belirtir. |
| Expression | Hesaplanan sütunlar (ing. Calculated Columns) için, değer hesaplamada kullanılmak istenen anlatımı belirtir. |
| MaxLength | Karakter dizi tipindeki sütunlar için sütuna atanabilecek en fazla karakter sayısını belirtir. |
| ReadOnly | Sütunun salt okunur olup olmadığını döndürür. |
| Table | Sütun nesnesinin içerisinde bulunduğu tablo nesnesini belirtir. |
| Unique | Sütun değerlerinin tekil olup olmadığını belirtir. |

Tablo 148, System.Data.DataColumn Sınıfı Üye Bilgileri

Satır Sınıfı

Veri tablosunda bir satırı simgeleyen System.Data.DataRow sınıfı üye bilgi ve açıklamaları aşağıda gösterilmiştir.

| Üye Adı | Açıklama |
| --- | --- |
| HasErrors | Satırın hata içerip içermediğini döndürür. |
| Item | Parametre olarak iletilen tamsayı, sütun adı veya sütun nesnesi ile ilgili alan değerini döndürür. |
| ItemArray | Tüm alan değerlerini nesne dizisi olarak almak veya nesneye iletmek amacıyla kullanılmalıdır. |
| RowError | Hata değerini karakter dizi olarak döndürür. |
| RowState | Satır durumunu DataRowState sayılabilir tipinde döndürür. |
| Table | Sütun nesnesinin içerisinde bulunduğu tablo nesnesini belirtir. |
| CancelEdit | Satır değerlerinde yapılan değişiklikleri iptal eder. |
| ClearErrors | Satır hatalarını temizler. |
| Delete | Satırı siler. |
| IsNull | Parametre olarak iletilen tamsayı, sütun adı veya sütun nesnesi ile ilgili alan değerinin boş olup olmadığını döndürür. |
| SetAdded | Satır durumunu eklenmiş olarak belirler. |
| SetModified | Satır durumunu değişmiş olarak belirler. |
| SetNull | Parametre olarak iletilen tamsayı, sütun adı veya sütun nesnesi ile ilgili alan değerini boşaltır. |

Tablo 149, System.Data.DataRow Sınıfı Üye Bilgileri

Satır durumu (ing. Row State), satır üzerinde yapılan düzenleme tipini göstermektedir.

| Üye Adı | Açıklama |
| --- | --- |
| Added | Satır yeni eklenmiştir. |
| Deleted | Satır silinmiştir. |
| Detached | Satır nesnesi oluşturulmuş ancak henüz listeye eklenmemiştir. |
| Modified | Satır değerleri değiştirilmiştir. |
| Unchanged | Satır değerlerinde herhangi bir değişiklik bulunmamaktadır. |

Tablo 150, System.Data.DataRowState Üye Bilgileri

DataRow.AcceptChanges metodu, satır üzerinde yapılan değişikliklerin kabul edilmesini sağlamakta, satır durumu Unchanged değerini almaktadır.

Veri Görünüm Sınıfı

Tablo verisinin farklı şekillerdeki görünümünü simgeleyen ve sıralama, filtreleme, arama gibi temel işlevleri sağlayan System.Data.DataView sınıfı üye bilgi ve açıklamaları aşağıda gösterilmiştir.

| Üye Adı | Açıklama |
| --- | --- |
| AllowDelete | Veri görünümünün silme işlemine izin verip vermediğini belirtir. |
| AllowEdit | Veri görünümünün güncelleme işlemine izin verip vermediğini belirtir. |
| AllowNew | Veri görünümünün ekleme işlemine izin verip vermediğini belirtir. |
| Count | Filtrelenmiş kayıt sayısını döndürür. |
| RowFilter | Filtreleme anlatım değerini belirtir. |
| Sort | Sıralama yapılmak istenen alanı belirtir. |
| Table | Veri görünümünün kaynağı olan tablo nesnesini döndürür. |
| AddNew | Veri görünümüne yeni bir kayıt ekler. |
| Close | Veri görünümünü kapatır. |
| Delete | Parametre olarak iletilen indeks numarasındaki kaydı siler. |
| Find | Sıralama yapılan alan üzerinden arama yapar. |
| GetEnumerator | IEnumerator arabirimini döndürür. |
| Open | Veri görünümünü açar. |
| ToTable | Veri tablosu nesnesi oluşturur ve veri görünümü kayıtları eklenmiş olarak oluşturulan tablo nesnesini döndürür. |

Tablo 151, System.Data.DataView Sınıfı Üye Bilgileri

Temel Veri İşlemleri

Veri kümeleri temel işlemleri ve kullanımları hakkında bu bölümde bilgiler sunulacaktır.

Veri Kümesine Tablolarının Eklenmesi

Aşağıdaki kod parçasında veri kümesi nesnesi oluşturulması ve tablo nesnelerinin eklenmesi gösterilmiştir.

C#

DataSet ds = **new** DataSet();

// Kisiler isimli yeni bir tablo ekle.

ds.Tables.Add("Kisiler");

// Sehirler isminde yeni bir tablo ekle.

DataTable table = **new** DataTable("Sehirler");

ds.Tables.Add(table);

VB.NET

**Dim** ds **As** DataSet = **New** DataSet()

' Kisiler isimli yeni bir tablo ekle.

ds.Tables.Add("Kisiler")

' Sehirler isminde yeni bir tablo ekle.

**Dim** table **As** DataTable = **New** DataTable("Sehirler")

ds.Tables.Add(table)

Kod 371, Veri Kümesi Nesnesi Oluşturma ve Tablolar Ekleme

Veri Tablolarının Doldurulması

Veri kümesi veya tablosunun doldurulması için DbDataAdapter.Fill metodu kullanılmalıdır.

C#

SqlCommand secKomutu = **new** SqlCommand("select \* from Kisi", conn);

SqlDataAdapter adapter = **new** SqlDataAdapter(secKomutu);

DataSet ds = **new** DataSet();

adapter.Fill(ds);

**foreach** (DataTable table **in** ds.Tables) {

Console.WriteLine("Tablo adı: {0}", table.TableName);

Console.WriteLine("Kayıt Sayısı: {0}", table.Rows.Count);

Console.WriteLine("Sütun Sayısı: {0}", table.Columns.Count);

}

VB.NET

**Dim** secKomutu **As** SqlCommand = **New** SqlCommand("**select** \* from Kisi", conn)

**Dim** adapter **As** SqlDataAdapter = **New** SqlDataAdapter(secKomutu)

**Dim** ds **As** DataSet = **New** DataSet()

adapter.Fill(ds)

**For** **Each** table **As** DataTable **In** ds.Tables

Console.WriteLine("Tablo adı: {0}", table.TableName)

Console.WriteLine("Kayıt Sayısı: {0}", table.Rows.Count)

Console.WriteLine("Sütun Sayısı: {0}", table.Columns.Count)

**Next**

Kod 372, Veri Kümesinin Doldurulması

Kod çıktısı aşağıda gösterilmiştir.

Tablo adı: Table

Kayıt Sayısı: 1

Sütun Sayısı: 5

DbDataReader.Fill metodu veri kümesi içerisine Table isminde yeni bir tablo eklemiş ve eklenen tablo kişi tablo verisi ile doldurulmuştur.

DbDataReader.Fill metoduna veri tablosu nesnesini parametre olarak iletmek mümkündür.

C#

DataSet ds = **new** DataSet();

DataTable table = ds.Tables.Add("Kisiler");

adapter.Fill(table);

VB.NET

**Dim** ds **As** DataSet = **New** DataSet

**Dim** table **As** DataTable = ds.Tables.Add("Kisiler")

adapter.Fill(table)

Kod 373, Veri Tablosunun Doldurulması

Veri küme ve tablolarının doldurulmasında DbDataReader nesneleri kullanılmaktadır.

Tablo Verilerine Erişim

Tablo verilerine erişim amacıyla DataTable.Rows ve DataTable.Columns özellikleri ile ifade edilen satır ve sütunlarda döngü içerisinde dolaşılabilir.

C#

**static** **void** TabloGoster(DataTable table) {

// Sütun isimlerini yazdır.

**for** (**int** i = 0; i < table.Columns.Count; i++)

Console.Write("{0}\t", table.Columns[i].ColumnName);

**if** (table.Rows.Count == 0)

Console.WriteLine("Kayıt bulunmamaktadır");

**else** {

// Satır ve sütunlarda dolaş.

**for** (**int** i = 0; i < table.Rows.Count; i++) {

**for** (**int** j = 0; j < table.Columns.Count; j++)

Console.Write("{0}\t", table.Rows[i][j]);

// Satır durumunu yazdır.

Console.WriteLine(table.Rows[i].RowState);

}

}

}

VB.NET

**Shared** **Sub** TabloGoster(**ByVal** table **As** DataTable)

' Sütun isimlerini yazdır.

**For** i **As** **Integer** = 0 **To** table.Columns.Count - 1

Console.Write("{0}" & vbTab, table.Columns(i).ColumnName)

**Next**

**If** table.Rows.Count = 0 **Then**

Console.WriteLine("Kayıt bulunmamaktadır")

**Else**

' Satır ve sütunlarda dolaş.

**For** i **As** **Integer** = 0 **To** table.Rows.Count - 1

**For** j **As** **Integer** = 0 **To** table.Columns.Count - 1

Console.Write("{0}" & vbTab, table.Rows(i)(j))

**Next**

' Satır durumunu yazdır.

Console.WriteLine(table.Rows[i].RowState)

**Next**

**End** **If**

**End Sub**

Kod 374, Tablo Verilerine Erişim

İçerisinde bir kayıt olan Kisi isimli tablo için kod çıktısı aşağıda gösterilmiştir.

Id TcKimlikNo Ad Soyad DogumTarih

1 59716221122 Ahmet Yılmaz 11.11.1955 Unchanged

Verinin Düzenlenmesi

Bağlantısız model gereği, veri tablosu satır değerleri üzerinde yapılan düzenlemeler veritabanına otomatik olarak yansıtılmamaktadır. Aşağıdaki kod parçasında tablo verileri üzerinde çeşitli düzenlemeler yapılması ve yapılan düzenlemelerin listelenmesi gösterilmiştir.

C#

// Orjinal veri ile tabloyu doldur.

adapter.Fill(table);

// Yeni bir satır ekle.

DataRow yeniSatir = table.NewRow();

yeniSatir["TcKimlikNo"] = 67453231234;

yeniSatir["Ad"] = "Mustafa";

yeniSatir["Soyad"] = "Yılmaz";

yeniSatir["DogumTarih"] = **new** DateTime(1954, 1, 1);

table.Rows.Add(yeniSatir);

// Satır değerlerini nesne dizisi olarak ekle.

**object** [] satir = **new** **object**[] {**null**, 77665599234, "Hasan", "Tok",   
 **new** DateTime(1988, 5, 4)};

table.Rows.Add(satir);

// İlk satırda değişiklikler yap.

DataRow row = table.Rows[0];

row["Ad"] = "Murat";

// Yapılan değişikliklerin bulunduğu tabloyu al.

DataTable degisiklikler = table.GetChanges();

TabloGoster(degisiklikler);

VB.NET

' Orjinal veri ile tabloyu doldur.

adapter.Fill(table)

' Yeni bir satır ekle.

**Dim** yeniSatir **As** DataRow = table.NewRow()

yeniSatir("TcKimlikNo") = 67453231234

yeniSatir("Ad") = "Mustafa"

yeniSatir("Soyad") = "Yılmaz"

yeniSatir("DogumTarih") = **New** DateTime(1954, 1, 1)

table.Rows.Add(yeniSatir)

' Satır değerlerini nesne dizisi olarak ekle.

**Dim** satir() **As** **Object** = {**Nothing**, 77665599234, "Hasan", "Tok", \_  
 **New** DateTime(1988, 5, 4)}

table.Rows.Add(satir)

' İlk satırda değişiklik yap.

**Dim** row **As** DataRow = table.Rows(0)

row("Ad") = "Murat"

' Yapılan değişikliklerin bulunduğu tabloyu al.

**Dim** degisiklikler **As** DataTable = table.GetChanges()

TabloGoster(degisiklikler)

Kod 375, Verinin Düzenlenmesi

Kod çıktısı aşağıda gösterilmiştir.

Id TcKimlikNo Ad Soyad DogumTarih

1 59716221122 Murat Yılmaz 11.11.1955 Modified

67453231234 Mustafa Yılmaz 01.01.1954 Added

77665599234 Hasan Tok 04.05.1988 Added

DataTable.AcceptChanges metodunun çağrılması ile tüm satırlar üzerinde yapılan değişiklikler kabul edilmiş olmakta ve satır durumları DataRowState.Unchanged değerini almaktadır. DataTable.RejectChanges metodu, satırlarda yapılan tüm değişikliklerin iptal edilmesini sağlamak için kullanılmalıdır.

Düzenlemelerin Kaydedilmesi

Veri kümeleri, tabloları veya satırlarda yapılan değişiklikler DbDataAdapter.Update metodu ile veritabanına iletilmelidir. Veri adaptörü tarafından yapılan işlem, her satırın durum değerine göre farklı veri komutlarını çalıştırmaktır. Örneğin satır yeni eklenmişse DbDataAdapter.InsertCommand veya satır değeri değişmişse DbDataAdapter.UpdateCommand komut nesnesi kullanılmaktadır.

C#

SqlCommand secKomutu = **new** SqlCommand("select \* from Kisi", conn);

// Ekleme, güncelleme ve silme için kullanılan komut nesneleri   
// oluşturulmaktadır. Her komut nesnesinin parametrelerinin eklenmesi  
// sadelik sağlamak amacıyla gözardı edilmiştir.

SqlCommand ekleKomutu = **new** SqlCommand("EkleKisi", conn);

ekleKomutu.CommandType = CommandType.StoredProcedure;

SqlCommand guncelleKomutu = **new** SqlCommand("GuncelleKisi", conn);

guncelleKomutu.CommandType = CommandType.StoredProcedure;

SqlCommand silKomutu = **new** SqlCommand("SilKisi");

silKomutu.CommandType = CommandType.StoredProcedure;

SqlDataAdapter adapter = **new** SqlDataAdapter(secKomutu);

// Adaptör komutları belirlenmektedir.

adapter.InsertCommand = ekleKomutu;

adapter.UpdateCommand = guncelleKomutu;

adapter.DeleteCommand = silKomutu;

// DbDataAdapter.Update metoduna içerisinde değişiklikler bulunan tablo   
// parametre olarak iletilmektedir.

adapter.Update(table);

// Düzenlemeler veritabanına iletildikten sonra değişiklikler kabul   
// edilmektedir.

table.AcceptChanges();

VB.NET

**Dim** secKomutu **As** SqlCommand = **New** SqlCommand("**select** \* from Kisi", conn)

' Ekleme, güncelleme ve silme için kullanılan komut nesneleri   
' oluşturulmaktadır. Her komut nesnesinin parametrelerinin eklenmesi  
' sadelik sağlamak amacıyla gözardı edilmiştir.

**Dim** ekleKomutu **As** SqlCommand = **New** SqlCommand("EkleKisi", conn)

ekleKomutu.CommandType = CommandType.StoredProcedure

**Dim** guncelleKomutu **As** SqlCommand = **New** SqlCommand("GuncelleKisi", conn)

guncelleKomutu.CommandType = CommandType.StoredProcedure

**Dim** silKomutu **As** SqlCommand = **New** SqlCommand("SilKisi")

silKomutu.CommandType = CommandType.StoredProcedure

**Dim** adapter **As** SqlDataAdapter = **New** SqlDataAdapter(secKomutu)

' Adaptör komutları belirlenmektedir.

adapter.InsertCommand = ekleKomutu

adapter.UpdateCommand = guncelleKomutu

adapter.DeleteCommand = silKomutu

' DbDataAdapter.Update metoduna içerisinde değişiklikler bulunan tablo   
' parametre olarak iletilmektedir.

adapter.Update(table)

' Düzenlemeler veritabanına iletildikten sonra değişiklikler kabul   
' edilmektedir.

table.AcceptChanges()

Kod 376, Düzenlenen Verinin Kaydedilmesi

Verinin Sıralanması

Veri görünümleri, veritabanında bulunan görünümlerin (ing. Database View) bağlantısız modelde karşılığı olarak değerlendirilebilir.

Aşağıdaki örnekte kişi tablosu verisinin veri görünümü yardımıyla Ad sütununa göre sıralanması gösterilmiştir.

C#

// Görünüm nesnesini oluştur.

DataView gorunum = **new** DataView(table, **null**, "Ad",   
 DataViewRowState.CurrentRows);

// Sıralı satırları göster.

**foreach** (DataRowView dr **in** gorunum)

Console.WriteLine("{0} {1}", dr["Ad"], dr["Soyad"]);

VB.NET

' Görünüm nesnesini oluştur.

**Dim** gorunum **As** DataView = **New** DataView(table, **Nothing**, "Ad", \_  
 DataViewRowState.CurrentRows)

' Sıralı satırları göster.

**For** Each dr **As** DataRowView **In** gorunum

Console.WriteLine("{0} {1}", dr("Ad"), dr("Soyad"))

**Next**

Kod 377, Verinin Sıralanması

Kod çıktısı aşağıda gösterilmiştir.

Hasan Tok

Murat Yılmaz

Mustafa Yılmaz

Arama Yapmak

Tablo verilerinde arama yapmak için kullanılabilecek iki model bulunmaktadır.

DataRow.Find metodu, tablo ana anahtar sütunlarında arama yapmak amacıyla kullanılabilmektedir. DataRow.Find metodunun kullanılabilmesi için ana anahtar sütunlarının DataTable.PrimaryKey özelliğine atanmış olmaları gerekmektedir.

C#

adapter.Fill(table);

table.PrimaryKey = **new** DataColumn[] { table.Columns["Id"] };

DataRow row = table.Rows.Find(2);

**if** (row != **null**)

Console.WriteLine("{0} {1}", row["Ad"], row["Soyad"]);

**else**

Console.WriteLine("2 numaralı kişi bulunamadı");

VB.NET

adapter.Fill(table)

table.PrimaryKey = **New** DataColumn() {table.Columns("Id")}

**Dim** row **As** DataRow = table.Rows.Find(2)

**If** **Not** row **Is** **Nothing** **Then**

Console.WriteLine("{0} {1}", row("Ad"), row("Soyad"))

**Else**

Console.WriteLine("2 numaralı kişi bulunamadı")

**End** **If**

Kod 378, Veri Tablo Nesneleriyle Arama Yapmak

Arama amacıyla kullanılabilecek bir diğer yöntem de veri görünümleridir. DataView.Find metodu, sıralama yapılmış olan alan üzerinde arama yapabilmektedir. Aşağıdaki örnekte TcKimlikNo alanında arama yapılması gösterilmiştir.

C#

DataView gorunum = **new** DataView(table, **null**, "TcKimlikNo",   
 DataViewRowState.CurrentRows);

**int** indeks = gorunum.Find(77665599234);

**if** (indeks < 0)

Console.WriteLine("Kayıt bulunamadı");

**else**

Console.WriteLine(gorunum[indeks]["Ad"]);

VB.NET

**Dim** gorunum **As** DataView = **New** DataView(table, **Nothing**, "TcKimlikNo", \_  
 DataViewRowState.CurrentRows)

**Dim** indeks **As** **Integer** = gorunum.Find(77665599234)

**If** indeks < 0 **Then**

Console.WriteLine("Kayıt bulunamadı")

**Else**

Console.WriteLine(gorunum(indeks)("Ad"))

**End** **If**

Kod 379, Veri Görünüm Nesneleriyle Arama Yapmak

Filtreleme Yapmak

Belirli kritere uyan satırların listelenmesi amacıyla, arama işleminde olduğu gibi, kullanılabilecek iki model bulunmaktadır.

DataTable.Select metoduna, filtrelemede kullanılmak istenen anlatım değeri iletilerek kritere uyan sütunlar elde edilebilir.

C#

**string** filtre = "Ad like 'M%'";

DataRow [] rows = table.Select(filtre);

**foreach** (DataRow row **in** rows)

Console.WriteLine("{0} {1}", row["TcKimlikNo"], row["Ad"]);

VB.NET

**Dim** filtre **As** **String** = "Ad like 'M%'"

**Dim** rows() **As** DataRow = table.Select(filtre)

**For** Each row **As** DataRow **In** rows

Console.WriteLine("{0} {1}", row("TcKimlikNo"), row("Ad"))

**Next**

Kod 380, Veri Tablosu Nesneleriyle Filtreleme Yapmak

Kod parçası ile, adı M ile başlayan satırlar elde edilmektedir. Aynı işlevin veri görünümleri ile sağlanması aşağıda gösterilmiştir.

C#

**string** filtre = "Ad like 'M%'";

DataView gorunum = **new** DataView(table, filtre, **null**, \_  
 DataViewRowState.CurrentRows);

**foreach** (DataRowView dr **in** gorunum)

Console.WriteLine("{0} {1}", dr["TcKimlikNo"], dr["Ad"]);

VB.NET

**Dim** filtre **As** **String** = "Ad like 'M%'"

**Dim** gorunum **As** DataView = **New** DataView(table, filtre, **Nothing**, DataViewRowState.CurrentRows)

**For** Each dr **As** DataRowView **In** gorunum

Console.WriteLine("{0} {1}", dr("TcKimlikNo"), dr("Ad"))

**Next**

Kod 381, Veri Görünüm Nesneleriyle Filtreleme Yapmak

Hesaplanan Alanlarla Çalışmak

DataColum.Expression özelliği alan değerinin hesaplanarak üretilmesi için kullanılan anlatım değerini belirtmektedir.

C#

DataColumn adSoyad = **new** DataColumn("AdSoyad", **typeof**(**string**),   
 "Ad + ' ' + Soyad");

table.Columns.Add(adSoyad);

**foreach**(DataRow row **in** table.Rows)

Console.WriteLine(row["AdSoyad"]);

VB.NET

**Dim** adSoyad **As** DataColumn = **New** DataColumn("AdSoyad", **GetType**(**String**), \_  
 "Ad + ' ' + Soyad")

table.Columns.Add(adSoyad)

**For** Each row **As** DataRow **In** table.Rows

Console.WriteLine(row("AdSoyad"))

**Next**

Kod 382, Hesaplanan Alan Kullanım Örneği

Kod çıktısı aşağıda gösterilmiştir.

Murat Yılmaz

Mustafa Yılmaz

Hasan Tok

DataColumn.Expression özelliği toplam, averaj gibi fonksiyonları da desteklemektedir. Kullanılabilecek fonksiyonlar hakkında bilgi almak için MSDN dokümantasyonunu inceleyiniz.

XML ve Veri Kümeleri

DataSet.WriteXml ve DataSet.ReadXml metotları, veri kümesi bilgilerinin XML yapısında dışa aktarılabilmesini ve XML yapısında olan verinin yüklenebilmesini sağlamaktadır.

Aşağıdaki örnekte bu metotların kullanımı gösterilmiştir.

C#

SqlCommand secKomutu = **new** SqlCommand("OkuKisi", conn);

secKomutu.CommandType = CommandType.StoredProcedure;

secKomutu.Parameters.Add("@Id", SqlDbType.Int);

secKomutu.Parameters[0].Value = 2;

SqlDataAdapter adapter = **new** SqlDataAdapter(secKomutu);

DataSet ds = **new** DataSet("Veriler");

ds.Tables.Add("Kisiler");

adapter.Fill(ds.Tables[0]);

ds.WriteXml(@"c:\kisiler.xml");

DataSet ds2 = **new** DataSet();

ds2.ReadXml(@"c:\kisiler.xml");

VB.NET

**Dim** secKomutu **As** SqlCommand = **New** SqlCommand("OkuKisi", conn)

secKomutu.CommandType = CommandType.StoredProcedure

**Dim** p **As** SqlParameter = **New** SqlParameter("@Id", SqlDbType.Int)

secKomutu.Parameters(0).Value = 2

**Dim** adapter **As** SqlDataAdapter = **New** SqlDataAdapter(secKomutu)

**Dim** ds **As** DataSet = **New** DataSet("Veriler")

ds.Tables.Add("Kisiler")

adapter.Fill(ds.Tables(0))

ds.WriteXml("c:\kisiler.xml")

**Dim** ds2 **As** DataSet = **New** DataSet()

ds2.ReadXml("c:\kisiler.xml")

Kod 383, Veri Kümeleri ve XML Kod Örneği

kisiler.xml isimli dosya içeriği aşağıda gösterilmiştir.

<?xml version="1.0" standalone="yes"?>

<Veriler>

<Kisiler>

<Id>2</Id>

<TcKimlikNo>67453231234</TcKimlikNo>

<Ad>Mustafa</Ad>

<Soyad>Yılmaz</Soyad>

<DogumTarih>1954-01-01T00:00:00+02:00</DogumTarih>

</Kisiler>

</Veriler>

Kullanım Önerileri

Geniş ve detaylı bir konu olan veri kümeleri konusunda temel kullanım yöntemleri ve sık yapılan işlemler hakkında verilen bilgilerin ardından, veri kümeleri kullanımında dikkat edilmesi önerilen bazı konular aşağıda listelenmiştir.

1. Veri kümelerinin amacı, az sayıda verinin bilgisayar hafızasında işlenebilmesidir. Tablo ve görünüm nesneleri tarafından sunulan çeşitli metotlarla bir çok işlemin kolaylıkla yapılması sağlansa da, büyük miktarda verinin yönetimi için veri kümeleri uygun değildir. Örneğin 100.000 kayıt içeren tablo verisini veri tablosuna doldurmak ve kayıtlar arasında filtreleme ve arama yapmak performansı olumsuz etkileyebilmektedir.
2. Veri kümeleri sadece ihtiyaç duyulduğunda kullanılmalıdır. Örneğin veritabanı tablosu verisinin çekilip metin dosyasına verilerin yazılması gibi işlemlerde direkt veri okuyucular kullanılmalıdır.
3. Veri kümeleri kullanımının standart olarak benimsendiği durumlarda, veri tablolarına yüklenen veri sayısının mutlaka sınırlanması önerilmektedir. Örneğin kullanıcının veri listelemesi yapabilmesi için bazı arama kriterlerini girmesini zorunlu tutmak gibi çeşitli önlemlerin alınması önerilmektedir.

Veri Sayfalama

Özellikle web tabanlı uygulama yazılımlarında en sık ihtiyaç duyulan işlevlerden birisi de verinin sayfalanarak kullanıcıya gösterilmesidir.

Veri sayfalamanın yapılması amacıyla temel iki yöntem kullanılabilmektedir.

İstemci Taraflı Sayfalama

İstemci tarafında yapılan sayfalama (ing. Client-Side Paging), verinin bütün olarak veritabanı sunucusundan çekilmesini ve çekilen verinin kullanıcıya sayfalanarak gösterilmesini ifade etmektedir.

Bu model aşağıdaki adımlardan oluşmaktadır.

1. Veritabanı sunucusundan herhangi bir sayfalama işlemi yapılmadan tüm veri çekilir.
2. Çekilen veri uygulama sunucuları hafızasında, örneğin veri kümeleri kullanılarak veya başka yöntemlerle saklanır.
3. Kullanıcının sayfa geçişlerinde veritabanı sunucusuna bağlanılmadan direkt veri kümesi içerisinden ilgili sayfa gösterilir.

Örneğin toplam kayıt sayısının 1.000, sayfa boyutunun 100 olması durumunda, 1.000 kayıt veritabanı sunucusundan çekilmekte ve uygulama sunucusu hafızasında saklanmaktadır. Kullanıcının 8. sayfayı istemesi durumunda, hafızada saklanan 800-900 arasındaki kayıtlar gösterilmektedir. Yöntem, veritabanı sunucu bağlantısının sadece bir kez yapılmasını sağlamaktadır.

Şekil 55, İstemci Taraflı Veri Sayfalama

İstemci taraflı veri sayfalama büyük kullanım kolaylığı sağlamasına rağmen aşağıdaki dezavantajları da beraberinde getirmektedir.

* Özellikle yüksek kayıt sayısına sahip tablolarda bu yöntemin tercih edilmesi ile, verinin tamamının tek seferde veritabanı sunucusundan çekilmesi sistem performansını olumsuz etkileyebilmektedir.
* Veri miktarının artması ile sayfalama işlemi yavaşlayabilmektedir.
* Kullanıcı tarafından görüntülenen tek sayfa olsa dahi tüm verinin veritabanı sunucusundan çekilmesi gerekmektedir.

Sunucu Taraflı Sayfalama

Sunucu taraflı sayfalama (ing. Server-Side Paging), sadece istenilen sayfayı oluşturan kayıtların veritabanı sunucusundan çekilebilmesini belirtmektedir.

Örneğin toplam kayıt sayısının 1.000, sayfa boyutunun 100 olması durumunda, kayıtlar veritabanından 100'erlik bölümler halinde çekilmekte ve kullanıcıya gösterilmektedir. Kullanıcının tüm sayfaları istemesi sonucu en fazla 10 kez veritabanı bağlantısı yapılmaktadır.

Şekil 56, Sunucu Taraflı Sayfalama

istemci taraflı sorgulamaya göre performans açısından büyük kazanımları olan bu yöntemin kullanımı daha karmaşıktır.

Yöntemin uygulanması için iki yaklaşım kullanılabilmektedir.

Tekil Değerler Kullanılarak Sayfalama

Tekil değerler kullanılarak sunucu taraflı veri sayfalaması aşağıdaki adımlardan oluşmaktadır.

1. Tekil değerin alabileceği en küçük değerden büyük kayıtların bulunduğu ilk sayfa tekil değer bazında sıralanarak veritabanı sunucusu tarafından döndürülür. Dönen sonuç kümesinin son kaydına ait tekil değer saklanır.
2. İkinci sayfa, saklanan tekil değerden büyük kayıtların bulunduğu sonuç kümesidir. Diğer sayfalar da aynı yöntem kullanılarak elde edilir.

Örneğin Kisi tablosunun aşağıda listelenen kayıtları içerdiği durumda sayfalama tekil bir alan olan Id üzerinden yapılabilir.

Id TcKimlikNo Ad Soyad DogumTarih

1 59716221122 Ahmet Yılmaz 11.11.1955 00:00:00

2 59716221123 Mustafa Yılmaz 11.11.1965 00:00:00

4 59716221124 Hasan Tok 11.11.1935 00:00:00

5 59716221125 Mehmet Durmaz 11.11.1955 00:00:00

7 59716221126 Ayşe Aydın 11.11.1985 00:00:00

11 59716221127 Fatma Orhan 11.11.1945 00:00:00

14 59716221128 Nuri Yılmaz 11.11.1975 00:00:00

Sayfa boyutunun 3 olduğu örnek senaryoda toplam 7 kaydın sayfalanması aşağıdaki adımları içermektedir.

1. En küçük değer olabilecek 0 değerinden büyük ilk üç kayıt Id alanı bazında sıralanarak veritabanı sunucusu tarafından döndürülmektedir. Dönen kayıtlar 1, 2 ve 4 numaralı kayıtlar olacaktır.   
   (1. Sayfa)
2. 4 değerinden büyük ilk üç kayıt sunucudan döndürülecektir. Dönen kayıtlar 5, 7, ve 11 numaralı kayıtlar olacaktır.   
   (2. Sayfa)
3. 11 değerinden büyük ilk üç kayıt sunucudan döndürülecektir. Sadece 14 numaralı kayıt elde edilecek ve son sayfaya ulaşılacaktır.   
   (3. Sayfa)

Yöntemin tablo indeksleri kullanımını temel alması ile çok büyük miktarda verinin en hızlı şekilde sayfalaması yapılabilmektedir.

Aşağıda yöntemin uygulanmasının gösterildiği SQL Server depolanmış prosedür örneği gösterilmiştir.

**Create** **Procedure** [AraKisiSayfalayarak]

(

@Ad nvarchar(25) = **null**, @Soyad nvarchar(25) = **null**,

@Anahtar int = 0, @SayfaBoyut int = 3

)

**AS**

**SET** ROWCOUNT @SayfaBoyut

**select** Id, TcKimlikNo, Ad, Soyad, DogumTarih **from** Kisi

**where**

(@Ad **IS** **NULL** **OR** Ad **LIKE** @Ad +'%') **AND**

(@Soyad **IS** **NULL** **OR** Soyad **LIKE** @Soyad +'%') **AND**

(Id > @Anahtar)

Kod 384, Tekil Değerler Kullanılarak Sayfalama Örneği

Depolanmış prosedür ile ad veya soyad alanlarında arama yapılabilmekte ve arama sonucu dönen kayıtlar sayfalanabilmektedir.

Veritabanı Sunucusu Yöntemleri

Veri sayfalamanın sunucu üzerinde yapılabilmesi amacıyla farklı veritabanı üreticileri çeşitli yöntemler sunmaktadır.

SQL Server 2005 sürümünden itibaren sunucu üzerinde sayfalama yapmak amacıyla aşağıdaki depolanmış prosedür örneği şablon olarak kullanılabilir. Diğer üreticiler tarafından sunulan çözümleri için ürün dokümantasyonunu inceleyiniz.

**Create** **Procedure** [AraKisiSayfalayarak]

(

@Ad nvarchar(25) = **Null**, @Soyad nvarchar(25) = **Null**,

@Siralama nvarchar(25) = 'Ad', @SatirBaslangic int = 1,

@SayfaBoyut int = 3

)

**AS**

**with** Liste **as** (**select** Row\_Number() **over** (

**Order** **By** **Case** @Siralama

**When** 'Ad' **Then** Ad

**End**,

**Case** @Siralama

**When** 'Ad DESC' **Then** Ad

**End** **Desc**,

**Case** @Siralama

**When** 'Soyad' **Then** Soyad

**End**,

**Case** @Siralama

**When** 'Soyad DESC' **Then** Soyad

**End** **Desc**

)

**AS** **Row**, Id, TcKimlikNo, Ad, Soyad, DogumTarih

**from** Kisi **where** (@Ad **is** **Null** **OR** Ad **Like** @Ad +'%') **AND**

(@Soyad **is** **Null** **OR** Soyad **Like** @Soyad +'%')

)

**Select** **Row**, Id, TcKimlikNo, Ad, Soyad, DogumTarih **from** Liste

**Where** **Row** **between** @SatirBaslangic **and** @SatirBaslangic+@SayfaBoyut-1;

Kod 385, Veritabanı Çözümleri Kullanılarak Sayfalama Örneği

Depolanmış prosedür, sayfalama desteği ile birlikte kayıtların sıralanmasını da sağlamaktadır. Örneğin   
AraKisiSayfalayarak(null, null, "Soyad DESC", 1, 2) şeklinde çağrım ile soyadı sırasında azalan ilk iki kayıt döndürülmektedir.

Bir insanın değeri bayağı kesire benzer; Pay gerçek değerini gösterir, payda kendisini ne zannettiğini. Paydanın değeri arttıkça kesrin değeri azalır.

Lev Tolstoy

Bölüm

25

İşlem Grubu Yönetimi

İ

şlem grupları (ing. Transactions), veri bütünlüğünü korumak amacıyla kullanılan yapılardır. Örneğin bir işlemin bütünüyle tamamlanabilmesi için farklı tablolarda veri değişiklikleri ve eklemeleri yapmak gerekebilir. Yapılan veri değişikliklerinin veritabanına bütünüyle yazılması işlemin başarıyla tamamlanması için önem taşımaktadır. Böyle bir durumda, herhangi bir tabloda hata oluşması ile, diğer tablolarda yapılan değişikliklerin de geri alınması gerekmektedir.

Temel Kavramlar

İşlem grubu yönetimi veritabanı yönetim sistemi tarafından yapılmaktadır ve işlem gruplarının yazılım performansına direkt etkisi bulunmaktadır. Gereksiz yere veya hatalı işlem grubu kullanımı sonucu performansın büyük ölçüde düşmesi, veritabanı üzerinde gereksiz kilitlerin oluşması ve tüm sistemin çalışamaz hale gelmesi mümkündür. Özellikle satır bazlı kayıt kilitleme (Row Level Locking) yeteneği olmayan veritabanı yönetim sistemlerinde işlem grupları kullanımında dikkatli olunması önerilmektedir.

Kayıt Kilitleme

Satır bazlı kilitleme, veritabanı yönetim sisteminin, işlem grubu içerisinde değeri değiştirilmiş, silinmiş veya eklenmiş olan bir kaydı, kayıt bazında yönetebilmesine verilen isimdir. Örneğin bir işlem grubu içerisinde tabloda bulunan kaydın değeri değiştirilebilir. İşlem grubu henüz sonuçlanmadan, bir başka kullanıcının bu kayda erişememesi, ancak aynı tablodaki diğer kayıtlara erişebilmesi gerekmektedir. Bu durumda, satır bazlı kilitleme yeteneği olan veritabanı yönetim sistemleri işlemi başarıyla yönetebilmelerine rağmen, sadece tablo veya blok bazlı kilitlemenin (ing. Table and Block Lock) bulunduğu veritabanı yönetim sistemleri diğer kullanıcıyı bekletebilmektedirler.

Başlatma ve Tamamlama

Farklı veri işlemleri içerebilen işlem grubunun başlatılması, tamamlanması ve iptal edilmesi sırasıyla Begin Transaction, Commit Transaction ve RollBack Transaction kavramlarıyla ifade edilmektedir.

İşlem grubu başlatılması ile veritabanı yönetim sistemine, o andan itibaren yapılan işlemlerin başlatılan işlem grubu içerisinde olacağı belirtilmektedir. İşlem grubunun tamamlanması ile işlem grubu içerisinde yapılan tüm veri işlemleri de tamamlanmaktadır. İşlem grubunun iptal edilmesi, başladığı andan itibaren veri üzerinde yapılan değişikliklerin de geri alınmasını sağlamaktadır.

1. İşlem grubunu başlat.
2. A isimli tabloya kayıt ekle.
3. B isimli tablodan kayıt sil.
4. C isimli tabloda güncellemeler yap.
5. İşlem grubunu tamamla.

Yukarıdaki örnekte 5 numaralı adımdan sonra tüm değişiklikler geçerli olmaktadır. Örneğin 4 numaralı adımda bir hata olması durumunda, işlem grubunun iptal edilmesiyle, A ve B isimli tablolarda yapılan değişiklikler de geri alınmaktadır.

İzolasyon Seviyesi

İşlem grubu izolasyon seviyesi (ing. Transaction Isolation Level) , işlem grubu tarafından yapılan veri değişikliklerinin, henüz işlem grubu tamamlanmadan diğer kullanıcılara nasıl yansıtacağını ifade eden kavramdır.

* Tamamlanmış okuma (ing. Read Comitted), kullanıcının başlattığı işlem grubu içerisinde yapılan değişikliklerin ancak işlem grubunun tamamlanması ardından diğer kullanıcılara gözükebilir olmasını ifade etmektedir.
* Kirli okuma (ing. Dirty Read), yapılan değişikliklerin işlem grubu tamamlanmadan diğer kullanıcılara gözükebilir olmasını ifade etmektedir.
* Tekrarlanamaz okuma (ing. Nonrepetable Read), bir işlem grubu içerisinde okunan kaydın, aynı işlem grubu içerisinde tekrar okunduğunda değerinin değişmiş olabileceğini ifade etmektedir.
* Fantom okuma (ing. Phantom Read), bir işlem grubu tarafından belirli kriterlere göre elde edilmiş kayıtların, aynı işlem grubu içerisinde tekrar seçildiğinde toplam sayısının değişebileceğini ifade etmektedir.

İzolasyon seviyelerinin performans ve veri bütünlüğüne önemli etkileri bulunmaktadır.

Yazılım Katmanları ve İşlem Grubu

Bilindiği gibi işlem grubunun temel amacı birden çok işlemin başarı veya başarısızlıkla sonuçlanmasını sağlamaktır. Bu kapsamda, işlem grubu yönetiminin yapılması için en uygun yazılım katmanı iş katmanıdır.

Örneğin iş katmanı içerisinde bulunan iş nesnesinin, bir iş mantığı gerçekleştirimini yapabilmek için birden çok tablo verisini değiştirmesi çoğunlukla olası bir senaryodur. Dolayısıyla iş nesnesi tarafından yapılan işlem tek olsa da, veritabanı açısından değerlendirildiğinde farklı veritabanı tabloları üzerinde düzenlemeler yapılmaktadır. Böyle bir senaryoda iş katmanında işlem grubunun başlatılması ve tamamlanması en uygun modeli oluşturmaktadır.

Yerel ve Dağıtık İşlem Grupları

Yerel işlem grubu (ing. Local Transaction), sadece bir veritabanı yönetim sistemi üzerinde yönetilen işlem grubuna verilen isimdir. Bazı durumlarda işlem grubunun farklı veritabanı yönetim sistemleri üzerinde etkili olabilmesi gerekebilmektedir.

Uygulama yazılımının farklı veritabanı yönetim sistemleri veya farklı veritabanları üzerinde yaptığı değişikliklerin aynı işlem grubu tarafından yönetilmesi dağıtık işlem grubuna (ing. Distributed Transaction) örnek olarak verilebilir.

Dağıtık işlem gruplarının yönetilmesi amacıyla hazır ürünler bulunmaktadır. Örneğin Microsoft Distributed Transaction Coordinator - MSDTC yazılımı bu amaçla kullanılabilmektedir.

Kullanım Önerileri

Uygulama yazılımlarında işlem grubu kullanılırken benimsenmesi gereken bazı öneriler aşağıda sunulmuştur.

1. Gereksiz yere işlem grubu başlatılmasının sistem performansına negatif etkileri olduğu unutulmamalıdır. Buna rağmen, veri bütünlüğünün korunması amacıyla gerekli her yerde işlem grubu kullanılmalıdır.
2. Sadece veri okuma işlemlerinde işlem grubu kullanımından kaçınmalıdır.
3. İşlem grubunun başlatılması ve tamamlanması / iptal edilmesi arasında geçen süre kritiktir. Yazılım geliştirirken, bu arada yapılan işlemlerin olabildiğince kısa sürmesi için gerekli önlemler alınmalıdır.

İşlem Grubu Kullanımı

İşlem grubu kullanımı ADO.NET mimarisi tarafından desteklenmekte ve iki temel çözüm sunulmaktadır.

* Yerel işlem grubu desteği
* Dağıtık işlem grubu desteği

Yerel İşlem Grubu Kullanımı

System.Data.Common.DbTransaction sınıfı temel yerel işlem grubu sınıfıdır.

| Üye Adı | Açıklama |
| --- | --- |
| Connection | İşlem grubunun içerisinde bulunduğu veritabanı bağlantı nesnesini belirtir. |
| IsolationLevel | İşlem grubu izolasyon seviyesini belirtir. |
| Commit | İşlem grubunu tamamlar. |
| Rollback | İşlem grubunu iptal eder. |

Tablo 152, System.Data.Common.DbTransaction Sınıfı Üye Bilgileri

Aşağıdaki örnek kod parçasında yeni bir işlem grubu oluşturulması ve komut nesnelerinin oluşturulan işlem grubu içerisinde çalıştırılması gösterilmiştir.

C#

// Yerel işlem gurubu nesnesi oluştur.

SqlTransaction t = conn.BeginTransaction();

// Komutlara içerisinde çalışmaları gereken işlem grubunu ilet.

SqlCommand cmd1 = **new** SqlCommand("EkleKullanici", conn, t);

SqlCommand cmd2 = **new** SqlCommand("SilAdres", conn, t);

. . .

**try** {

cmd1.ExecuteNonQuery();

cmd2.ExecuteNonQuery();

// İşlem grubunu tamamla.

t.Commit();

}

**catch** (Exception exc) {

// Hata oluşması durumunda işlem grubunu iptal et.

t.Rollback();

}

**finally** {

// Her durumda veritabanı bağlantısını kapat.

conn.Close();

}

VB.NET

' Yerel işlem gurubu nesnesi oluştur.

**Dim** t **As** SqlTransaction = conn.BeginTransaction()

' Komutlara içerisinde çalışmaları gereken işlem grubunu ilet.

**Dim** cmd1 **As** SqlCommand = **New** SqlCommand("EkleKullanici", conn, t)

**Dim** cmd2 **As** SqlCommand = **New** SqlCommand("SilAdres", conn, t)

. . .

**Try**

cmd1.ExecuteNonQuery()

cmd2.ExecuteNonQuery()

' İşlem grubunu tamamla.

t.Commit()

**Catch** exc **As** Exception

' Hata oluşması durumunda işlem grubunu iptal et.

t.Rollback()

**Finally**

**'** Her durumda veritabanı bağlantısını kapat.

conn.Close()

**End** **Try**

Kod 386, Yerel İşlem Grubu Örneği

İşlem grubu kaydetme noktası (ing. Transaction Savepoint), aynı işlem grubu içerisinde kaydedilen herhangi bir noktaya dönebilmeyi sağlamaktadır. İşlem grubu kaydetme noktası tüm .NET veri sağlayıcıları tarafından desteklenmemektedir.

Aşağıdaki örnekte SQL Server .NET veri sağlayıcısı ile kullanım gösterilmiştir.

C#

SqlTransaction t = conn.BeginTransaction();

SqlCommand cmd1 = **new** SqlCommand("EkleKullanici", conn, t);

SqlCommand cmd2 = **new** SqlCommand("SilAdres", conn, t);

SqlCommand cmd3 = **new** SqlCommand("SilSehir", conn, t);

. . .

**try** {

cmd1.ExecuteNonQuery();

cmd2.ExecuteNonQuery();

// İşlem grubu kaydetme noktası oluştur.

t.Save("BirinciNokta");

**try** {

cmd3.ExecuteNonQuery();

. . .

}

**catch** (Exception exc) {

// Oluşturulan kaydetme noktasına kadar yapılan işlemleri iptal et.

t.Rollback("BirinciNokta");

}

}

**catch** (Exception exc) {

t.Rollback();

}

VB.NET

**Dim** t **As** SqlTransaction = conn.BeginTransaction()

**Dim** cmd1 **As** SqlCommand = **New** SqlCommand("EkleKullanici", conn, t)

**Dim** cmd2 **As** SqlCommand = **New** SqlCommand("SilAdres", conn, t)

**Dim** cmd3 **As** SqlCommand = **New** SqlCommand("SilSehir", conn, t)

. . .

**Try**

cmd1.ExecuteNonQuery()

cmd2.ExecuteNonQuery()

' İşlem grubu kaydetme noktası oluştur.

t.Save("BirinciNokta")

**Try**

cmd3.ExecuteNonQuery()

. . .

**Catch** exc **As** Exception

**'** Oluşturulan kaydetme noktasına kadar yapılan işlemleri iptal et.

t.Rollback("BirinciNokta")

**End** **Try**

**Catch** exc **As** Exception

t.Rollback()

**End** **Try**

Kod 387, İşlem Grubu Kaydetme Noktası Kullanım Örneği

Dağıtık İşlem Grubu Kullanımı

Dağıtık işlem grupları bir veya daha fazla veritabanı üzerinde işlem grubu yönetimi yapabilmeyi sağlamaktadır. Bu konu başlığında, dağıtık işlem grupları ile birlikte MSDTC yazılımının kullanımı hakkında bilgiler sunulacaktır.

MSDTC

MSDTC , Microsoft tarafından hazırlanmış ve dağıtık işlem grupları yönetimini yapabilen yazılımdır. MSDTC ile veritabanı yönetim sistemleri dışındaki farklı ürünler üzerinde de işlem grubu uygulanabilmektedir.

Bir hizmet uygulaması olarak çalışan MSDTC ve System.Transactions.dll isimli Assembly içerisinde bulunan sınıflar yardımıyla işlem gruplarının kolaylıkla yönetimi yapılabilmektedir.

Ekran Çıktısı 92, MSDTC Hizmeti

MSDTC konfigürasyonunu yapmak amacıyla Kontrol Panel | Yönetimsel Araçlar | Bileşen Hizmetleri (Control Panel | Administrative Tools | Component Services) kısa yolu ile çalıştırılabilen bileşen hizmetleri uygulaması kullanılmalıdır.

Ekran Çıktısı 93, Bileşen Hizmetleri Uygulaması ve MSDTC

Bileşen hizmetleri penceresi ile aktif işlem grupları hakkında bilgiler ve çeşitli istatistikler elde edilebilmektedir.

Yapılandırılmak edilmek istenen bilgisayar simgesi üzerinde sağ tıklanarak özellikler (Properties) komutunun verilmesi ile konfigürasyon penceresi açılmaktadır.

Ekran Çıktısı 94, MSTDC Konfigürasyon Penceresi

Yeni İşlem Grubu Başlatma ve Tamamlama

System.Transactions.TransactionScope sınıfı, bulunduğu kod bloğu içerisinde yapılan tüm veritabanı işlemlerinde aynı işlem grubunun kullanılmasını sağlamaktadır. IDisposable arabirim gerçekleştirimini yapan sınıf ile, TransactionScope nesnesinin oluşturulması ve yok edilmesi arasında bulunan kod bloğu içerisinde işlem grubu geçerli olmaktadır.

C#

// TransactionScope nesnesi oluştur.

**using** (TransactionScope txc = **new** TransactionScope()) {

// Blok içerisinde yapılan tüm işlemler işlem grubunun parçası   
 // olacaktır.

**using** (SqlConnection conn1 = **new** SqlConnection("...")) {

conn1.Open();

// . . .

}

**using** (SqlConnection conn2 = **new** SqlConnection("...")) {

conn2.Open();

// . . .

}

**using** (OracleConnection conn3 = **new** OracleConnection("...")) {

conn3.Open();

// . . .

}

// İşlem grubunu tamamla.

txc.Complete();

}

VB.NET

' TransactionScope nesnesi oluştur.

**Using** txc **As** TransactionScope = **New** TransactionScope()

' Blok içerisinde yapılan tüm işlemler işlem grubunun parçası   
 ' olacaktır.

**Using** conn1 **As** SqlConnection = **New** SqlConnection("...")

conn1.Open()

' . . .

**End** **Using**

**Using** conn2 **As** SqlConnection = **New** SqlConnection("...")

conn2.Open()

' . . .

**End** **Using**

**Using** conn3 **As** OracleConnection = **New** OracleConnection("...")

conn3.Open()

' . . .

**End** **Using**

' İşlem grubunu tamamla.

txc.Complete()

**End** **Using**

Kod 388, Dağıtık Yeni İşlem Grubu Oluşturma

Kod parçasında using bloğu kullanılarak, bu blok içerisinde yapılan tüm işlemlerin txc isimli işlem grubu içerisinde olması sağlanmıştır. TransactionScope sınıfı IDisposable arabirim gerçekleştirimini yaptığı için, blok içerisinde olabilecek bir istisna otomatik olarak TransactionScope.Dispose metodunu tetikleyecek ve işlem grubu iptal edilecektir. Böylece, yerel işlem grubu yönetiminde olduğu gibi, istisnanın yakalanması ve işlem grubunun iptal edilmesi gerekmemektedir.

Aktif İşlem Grubuna Erişim

System.Transactions.Transaction sınıfı işlem grubunu simgeleyen sınıftır.

| Üye Adı | Açıklama |
| --- | --- |
| Current | Aktif işlem grubu nesnesini belirten statik özelliktir. |
| IsolationLevel | İşlem grubu izolasyon seviyesini belirtir özelliktir. |
| TransactionInformation | İşlem grubu hakkında daha detaylı bilgiye erişmek için kullanılması gereken özelliktir. |
| Clone | İşlem grubunun kopyasını döndürür. |
| Rollback | İşlem grubunu iptal eder. |

Tablo 153, System.Transactions.Transaction Sınıfı Üye Bilgileri

Kullanım Önerileri

System.Transactions ad alanı işlem grubunu belirli bir düzen içerisinde yönetebilmeyi sağlayan sınıflar içermektedir. Sunulan sınıflar ve mimari rahatlıkla tek veritabanı kullanan uygulama yazılımlarında da tercih edilebilmektedir. Özellikle iş katmanı tarafından yapılan işlem grubu yönetiminde TransactionScope sınıfı büyük kazanımlar sağlamaktadır.

İnsanoğlu, bilgeliği sevenler siyasi gücü ellerine alana kadar veya siyasi gücü ellerinde tutanlar bilgeliği sevene kadar problemlerin bittiğini görmeyecek.

Eflatun

Bölüm

26

Veritabanı Uygulamaları

A

DO.NET tarafından sunulan mimari ve sınıflar veri işlemlerinin belirli bir disiplin içerisinde yapılmasını sağlıyor olsa da yoğun programlama gerektirmektedir. Microsoft Visual Studio.NET, sunmuş olduğu görsel tasarım olanakları ile programcı tarafından geliştirilmesi gereken kaynak kodun büyük bölümünü otomatik olarak üretebilmekte, böylece zaman ve emekten tasarruf sağlanmaktadır.

Her konuda olduğu gibi, VS.NET ile yapılan görsel veritabanı programlama konusunda da, programcının VS.NET tarafından yapılan işlemlerin detaylarını, mantığını ve mimarisini biliyor olması büyük önem taşımaktadır.

Konu anlatımında, örneklemeler ve ekran çıktılarında SQL Server veritabanı kullanılmıştır. Buna rağmen, konunun okuyucu tarafından anlaşılmasıyla diğer .NET veri sağlayıcıları ile de kolaylıkla aynı işlemlerin yapılabileceği değerlendirilmektedir.

Genel Mimari

VS.NET ile veritabanı programlamanın temelini veri kümeleri oluşturmaktadır. Bilindiği gibi veri kümeleri çok katmanlı yazılım mimarisinde iş varlıklarına karşılık gelmekte ve her veritabanı uygulamasında iş varlıkları olarak veri kümelerinin kullanım zorunluluğu bulunmamaktadır.

Kitapta şu ana kadar aktarılan konularda Untyped Dataset olarak tanımlanan veri kümeleri kullanılmıştır. Bu kavram ile anlatılmak istenen, veri kümesi, tabloları, sütun ve satırlarının yüklenen veriye göre çalışma zamanında dinamik olarak oluşturulmasıdır.

Bir diğer yaklaşım Typed Dataset kavramıdır. Bu yaklaşım, otomatik olarak oluşturulan sınıflar ile, veri yapısının kaynak koda statik olarak yansımasını sağlamaktadır. Böylece kaynak kod içerisinde tablolara ve alanlara isimleriyle erişmek mümkün olmaktadır.

Her iki yaklaşımın da farklı kazanımları bulunsa da, özellikle ihtiyaç olmadığı durumlarda, gerek kodlama hata olasılığını azaltmak gerek nesne yönelimli programlama kavramını olabildiğince veri katmanına yansıtmak amacıyla Typed Dataset kullanımının benimsenmesi önerilmektedir.

Tablo Adaptörleri ve Sorgular

VS.NET ile yapılan görsel veritabanı programlama dört yapının kullanımını içermektedir.

* Veri Kümeleri
* Veri Tabloları
* Tablo Adaptörleri
* Sorgular

Tablo adaptörleri (ing. Table Adapter) olarak isimlendiren sınıflar, sağlamış oldukları sorgu metotları ile veriye erişim sağlayan sınıflardır. Bu sınıflar arka planda, kitapta daha önceki bölümlerde aktarılan veri komutları, okuyucu ve adaptörlerini kullanmaktadırlar.

Aşağıda VS.NET veri kümesi editörü ve bu kavramlar gösterilmiştir.

Şekil 57, VS.NET Veri Kümesi Editörü

Sorgu (ing. Query) kavramından sadece veri listeleme veya filtreleme amacıyla kullanılan metotlar anlaşılmamalıdır. Sorgular verinin silinmesi veya güncellenmesi gibi işlemleri de yapabilmektedir.

XSD Dosyaları ve Veri Kümesi Editörü

XSD (Xml Schema Definition) dosyaları veri yapısının tanımlandığı dosyalardır. Aşağıdaki ekran çıktısında VS.NET XML Schema Editor tarafından açılan örnek XSD dosyası gösterilmiştir.

Ekran Çıktısı 95, XSD Editörü

Veri kümeleri ve tablo yapıları, tabloların birbirleriyle olan ilişkileri bu dosya içerisinde saklanmaktadır. Aşağıdaki ekran çıktısında veri kümesi bilgilerinin saklandığı dosya özellikleri bulunmaktadır.

Ekran Çıktısı 96, XSD Veri Kümesi Dosya Özellikleri

Custom Tool olarak MSDataSetGenerator değerinin bulunduğuna dikkat ediniz. VS.NET tarafından yapılan işlem, XSD dosya içeriğine göre otomatik olarak C# ve VB.NET sınıflarının oluşturulmasıdır. Böylece programcı görsel olarak veri kümesi ve tablolarını tasarlamakta, yapılan görsel düzenlemeler XSD yapısına dönüştürülmektedir. Son adım olarak MSDataSetGenerator otomatik kod üreticisi yardımıyla XSD içeriğinden kaynak kod oluşturulmaktadır.

VS.NET esnek yapısı ile programcıların da kod üreticileri geliştirmeleri ve VS.NET içerisinden kullanmaları mümkündür.

Veri Kümeleri Kullanımı

VS.NET ile veritabanı programlama konusunda ilk adım veri kümelerinin oluşturulmasıdır. Bunun için projeye veri kümesi eklenmelidir.

Ekran Çıktısı 97, Projeye Veri Kümesi Eklenmesi

Aşağıda gösterilen ekran çıktısında VS.NET veri editörü ile açılan boş veri kümesi ve özellikler penceresi birlikte gösterilmiştir.

Ekran Çıktısı 98, Boş Veri Kümesi ve Özellikler Penceresi

Veri kümesi özelliklerinin görsel olarak düzenlenebilmesi için özellikler penceresi kullanılabilmektedir. Özellik penceresinde bulunan değerlerin System.Data.DataSet sınıfı özellikleridir.

Projeye yeni veri kümesi eklenmesiyle VS.NET tarafından otomatik olarak oluşturulan dosyalar aşağıda gösterilmiştir. Genel yaklaşımın, kitabın daha önceki bölümlerinde aktarılan konfigürasyon ayarları editörü ve ayar sınıfı ile aynı olduğuna dikkat ediniz.

Ekran Çıktısı 99, Veri Kümesi Dosyaları

Kaynak Kod Üretimi

VS.NET tarafından üretilen kaynak kodun bulunduğu dosyalar C# ve VB.NET için sırasıyla Veriler.Designer.cs ve Veriler.Designer.vb isimli dosyalardır. Bu dosyaların içeriği, yapılan görsel düzenlemeler sonucunda VS.NET tarafından belirlenmektedir ve programcıların dosya içeriklerinde yapacakları değişiklikler kaybolmaktadır.

Yapılan görsel düzenlemeler otomatik olarak anında veya projenin kaydedilmesiyle bu dosyalara yansıtılmaktadır. Buna rağmen, bazen otomatik kod üretiminin yapılmaması durumunda, çözüm penceresinde veri kümesi dosyası üzerinde sağ tıklanarak Run Custom Tool komutu verilmesi gerekebilmektedir.

Aşağıda yeni eklenen veri kümesi için otomatik olarak oluşturulan veri kümesi sınıf tanımı gösterilmiştir.

C#

**public** **partial** **class** Veriler : System.Data.DataSet {

. . .

}

VB.NET

**Partial** **Public** **Class** Veriler

**Inherits** System.Data.DataSet

. . .

**End** **Class**

Kod 389, VS.NET Tarafından Oluşturulan Veri Kümesi Sınıfı

Yapılan işlem DataSet sınıfını miras alan kısmi Veriler isimli sınıfın oluşturulmasıdır.

Sınıfın kısmi olarak tanımlanması ile, programcıların bir başka kaynak kod dosyasında aynı isimli sınıfı tanımlamaları ve farklı üyeler ekleyebilmeleri mümkün olmaktadır. Bunun için kullanılabilecek en pratik yöntem veri kümesi editörü üzerinde sağ tıklayarak View Code komutunun verilmesidir.

Ekran Çıktısı 100, Veri Kümesi Editörü View Code Komutu

Komutun verilmesiyle birlikte veriler.cs ve veriler.vb isimli dosyalar projede bulunmuyorsa oluşturulmakta, bulunuyorsa açılmaktadır.

Ekran Çıktısı 101, Programcı Tanımlı Veri Kümesi Sınıf Dosyaları

Programlama sırasında sık duyulan gereksinimlerden birisi de, otomatik kod üretimi ile oluşturulan sınıfların genişletilebilmesidir. Yukarıdaki yöntem ile otomatik oluşturulan Veriler isimli sınıfın kolaylıkla genişletilebilmesi mümkündür.

Aşağıda örnek kullanım gösterilmiştir.

C#

**partial** **class** Veriler {

**public** **void** OrnekMetot() {

}

}

VB.NET

**Partial** **Class** Veriler

**Public** **Sub** OrnekMetot()

**End** **Sub**

**End** **Class**

Kod 390, Programcı Tanımlı Veri Kümesi Sınıfı

Böylece Veriler isimli sınıf, görsel düzenlemeler sonucu otomatik olarak oluşturulan işlevlere ek olarak programcının eklediği üyeleri de içerebilmektedir.

Aktarılan bu yöntem görsel tasarım sonucu üretilen tüm sınıfların genişletilmesi amacıyla rahatlıkla kullanılabilir ve veri kümelerine özgü değildir.   
  
Otomatik üretilen sınıfı genişletmenin bir diğer yöntemi de, sınıfı miras alan yeni bir sınıf tanımlamaktır. Yazılım içerisinde yeni tanımlanan sınıf kullanımının benimsenmesi ile otomatik kod üretim kazanımları ve genişleyebilirlik birleştirilebilmektedir.

Tablo ve Adaptör Kullanımı

Görsel veritabanı programlamanın üç önemli parçası olan veri tabloları, adaptörleri ve sorgular hakkında bu konu başlığında açıklamalarda bulunulacaktır.

Veri Erişiminin Sağlanması

Veri kümesi editörü üzerinde sağ tıklanarak Add Table Adapter komutu verilerek tablo adaptör sihirbazı başlatılabilir.

Ekran Çıktısı 102, Veri Adaptörü Eklenmesi

Veri adaptörünün eklenme sürecinde veri tablosu ve veri tablosu yapısını belirleyen sorgu oluşturulmaktadır.

Veritabanı Bağlantısının Seçilmesi

Veri adaptör sihirbazında öncelikle veritabanı bağlantısı seçilmelidir.

Ekran Çıktısı 103, Veritabanı Bağlantı Dizesinin Belirtilmesi

Sihirbaz sayfasında listelenen veritabanı bağlantıları VS.NET sunucu gezgini içerisinde bulunan veritabanı bağlantılarıdır. New Connection komutu verilerek yeni bağlantı oluşturulabilmektedir.

No seçeneği, veritabanı bağlantı dizesine gizliliği olan kullanıcı adı ve şifresi değerlerinin yazılmasını engellemek amacıyla kullanılmaktadır.

Veritabanı Bağlantı Dizesinin Kaydedilmesi

Seçilen veritabanı bağlantı dizesi uygulama ayarları dosyasında saklanabilmektedir.

Ekran Çıktısı 104, Veritabanı Bağlantı Dizesinin Kaydedilmesi

Bir önceki sihirbaz sayfasında yapılan seçime göre, veritabanı bağlantı dizesi içerisinde gizliliği olan bilgiler depolanmamaktadır. Burada kullanılabilecek bir yöntem, önceki sayfada Yes seçeneği ile gizli değerlerin de saklanması ve kitabın konfigürasyon yönetimi bölümünde aktarılan bilgileri kullanarak bağlantı dizesinin korunmasıdır.

Ekran çıktısında görülen KitapConnectionString değeri veritabanı bağlantı dizesi adını ifade etmektedir.

Komut Tipinin Seçilmesi

Sihirbazın tamamlanması ile oluşturulacak veri tablo yapısının belirlenmesi amacıyla, bu işlemde kullanılacak komut tipi seçimi yapılmalıdır.

Komut tipi olarak iki değer belirtilebilmektedir.

* SQL İfadeleri
* Depolanmış prosedürler

Create New Stored Procedures seçeneği ile depolanmış prosedürler sihirbaz yardımıyla otomatik olarak oluşturulabilmektedir.

Ekran Çıktısı 105, Komut Tipinin Seçilmesi

Use SQL Statements seçeneği ile sonuç döndüren herhangi SQL cümlesi belirtilebilmektedir. Buna rağmen, kitabın daha önceki bölümlerinde sunulan öneriler çerçevesinde veri erişiminin sadece depolanmış prosedürler aracılığıyla yapılmasının en doğru çözümü oluşturduğu değerlendirilmektedir.

Depolanmış Prosedürlerin Seçimi

Use existing stored procedures seçeneğinin işaretlenmesi sonucu, verinin okunması, ekleme, silme ve güncelleme işlemleri için kullanılması istenen prosedürler bu sayfa yardımıyla belirtilmektedir.

Ekran Çıktısı 106, Depolanmış Prosedürlerin Seçimi

Ekran çıktısında, detayları kitabın daha önceki bölümlerinde aktarılan depolanmış prosedürler dört temel işlemin (Create – Read – Update – Delete, CRUD) yapılması amacıyla belirtilmiştir. Sihirbaz tarafından oluşturulacak tablo veri yapısını, verinin seçilmesi (select) için kullanılan depolanmış prosedürün döndürdüğü değerler belirlemektedir.

Adaptör Metotlarının İsimlendirilmesi

Verinin okunması amacıyla kullanılmak istenen tablo adaptör sınıfı metot isimleri bir sonraki sihirbaz sayfasında belirtilmektedir.

Ekran Çıktısı 107, Tablo Adaptör Sınıfı Metot İsimlerinin Belirtilmesi

Fill a DataTable seçeneği, parametre olarak alınan veri kümesinin OkuKisi isimli prosedür ile doldurulmasını sağlayacak metodun otomatik olarak oluşturulmasını sağlamaktadır.

Return a DataTable seçeneği, OkuKisi isimli prosedürün çağrılması ile doldurulan veri tablosunu döndüren metodun oluşturulmasını sağlamaktadır.

Sihirbazın Tamamlanması

Sihirbazın başarıyla tamamlanması ile otomatik kod üretimi yapılmakta ve belirtilen değerlere göre veri tablosu, tablo adaptörü ve sorgu oluşturulmaktadır.

Ekran Çıktısı 108, Tablo Adaptör Sihirbazının Tamamlanması

Oluşturulan veri tablosu, tablo adaptörü ve sorgu aşağıda gösterilmiştir.

Ekran Çıktısı 109, Otomatik Oluşturulan Yapılar

Verinin Görüntülenmesi

Verinin görüntülenmesi için tablo veya sorgu üzerinde sağ tıklayarak Preview Data … komutu verilmelidir.

Ekran Çıktısı 110, Verinin Tasarım Anında VS.NET ile Görüntülenmesi

Komutun verilmesiyle, veri tablosu yapısını belirleyen OkuKisi isimli prosedürün çağrılabilmesi amacıyla veri görüntüleme penceresi açılmaktadır.

Ekran Çıktısı 111, Veri Görüntüleme Penceresi

Tablo İsminin Düzenlenmesi

Tablo yapısını belirleyen depolanmış prosedür adı, otomatik olarak tablo adı olarak atanmaktadır. OkuKisi prosedürünün birebir Kisi tablo yapısını döndürdüğü değerlendirildiğinde tablo isminin Kisi olarak değiştirilmesi doğru bir yaklaşım olacaktır.

Ekran Çıktısı 112, Veri Tablo Adının Düzenlenmesi

Sınıf Yapısı

VS.NET tarafından oluşturulan sınıfların bulunduğu diyagram aşağıda gösterilmiştir.

Ekran Çıktısı 113, VS.NET Tarafından Üretilen Sınıflar

1. DataTable sınıfını miras alan KisiDataTable isimli sınıf üretilmiştir.
2. Kisi tablosundaki bir satırı simgeleyen ve DataRow sınıfını miras alan KisiRow sınıfını üretilmiştir.
3. Kisi tablosundaki her bir sütun için DataColumn sınıfını miras alan sütun sınıfları üretilmiştir.
4. İçerisinde Kisi tablosunda CRUD işlemlerinin yapılmasını sağlayan metotların bulunduğu KisiTableAdapter isimli sınıf üretilmiştir.

Tablo Adaptörleri Kullanımı

Tablo adaptörleri, içerdikleri metotlar ile verinin elde edilmesi ve veri işlemlerinin kolaylıkla yapılabilmesini sağlamaktadırlar. Tablo adaptörleri tarafından sağlanan temel işlev, farklı komut nesnelerinin yönetiminin görsel olarak yapılan tasarım sonucu otomatik olarak yapılmasıdır. Bu kapsamda tablo adaptörleri veri komutlarının toplandığı kapsayıcılar olarak değerlendirilebilir.

Tablo Adaptör Özelliklerinin Düzenlenmesi

VS.NET veri kümesi editöründe, tablo adaptörünün seçilmesi ile özellikler penceresinde görüntülenen değerler aşağıda gösterilmiştir.

Ekran Çıktısı 114, Tablo Adaptörü ve VS.NET Özellikler Penceresi

Özellikler penceresi ile yapılan düzenlemeler sonucu VS.NET tarafından kod üretimi tekrarlanmaktadır. Örneğin tablo adaptör sınıfının ön tanımlı ata sınıfı olan ve BaseClass özelliği ile belirtilen System.ComponentModel.Component değerinin değiştirilmesiyle tablo adaptör sınıfının miras aldığı sınıf da kaynak kod dosyası içerisinde değiştirilmektedir.

Tablo adaptörü özellikleri ve açıklamaları aşağıda gösterilmiştir.

| Özellik | Açıklama |
| --- | --- |
| BaseClass | Tablo adaptör sınıfının miras aldığı sınıfı belirtir. |
| Connection | Adaptör tarafından kullanılmak istenen veritabanı bağlantı değerini belirtir. |
| ConnectionModifier | Tablo adaptörünün kullandığı veritabanı bağlantı nesnesi adaptör sınıfının bir özelliği olarak oluşturulmaktadır. Bu değer, özellik giriş niteleyicisini (ing. Access Modifier) belirtmektedir. |
| DeleteCommand | Kayıt silme amacıyla kullanılan komut nesnesi özelliklerini belirtmektedir. |
| GenerateDBDirectMethods | True değerinin atanması durumunda kayıt silme, güncelleme ve ekleme için sırasıyla Delete, Update ve Insert isimli sınıf metotları oluşturulmaktadır. Bu metotlar içerisinde sırasıyla DeleteCommand, UpdateCommand ve InsertCommand ile ifade edilen komut nesneleri kullanılmaktadır. |
| InsertCommand | Kayıt ekleme amacıyla kullanılan komut nesnesi özelliklerini belirtmektedir. |
| Modifier | Oluşturulan adaptör sınıfı giriş niteleyicisini belirtmektedir. |
| Name | Sınıf adını belirtmektedir. |
| SelectCommand | Kayıt seçme amacıyla kullanılan komut nesnesi özelliklerini belirtmektedir. |
| UpdateCommand | Kayıt güncelleme amacıyla kullanılan komut nesnesi özelliklerini belirtmektedir. |

Tablo 154, Tablo Adaptörü Özellikleri ve Açıklamaları

Komut Özelliklerinin Düzenlenmesi

Tablo adaptörü sihirbazı ile oluşturulan komutların daha sonra özellikler penceresi ile düzenlenmesi mümkündür.

Ekran Çıktısı 115, Tablo Adaptörü Komut Özellikleri

Her bir komut, komut metni (CommandText), komut tipi (CommandType) ve parametre (Parameters) değerlerinden oluşmaktadır. Özellikler penceresi ile yapılan düzenlemeler, kaynak kod içerisinde bulunan DbCommand nesneleri özelliklerine otomatik olarak yansıtılmaktadır.

Sorgu Özelliklerinin Düzenlenmesi

Sorgular, tablo adaptör sınıfının metotlarına karşılık gelen yapılardır. Bir tablo adaptörü içerisinde birden çok sorgu bulunabilmektedir.

Mevcut sorgunun VS.NET veri kümesi editöründe seçilmesiyle özellikleri görüntülenmektedir.

Ekran Çıktısı 116, Sorgu Özellikleri

Her bir sorgu, üretilen tablo adaptör sınıfında en fazla iki metoda karşılık gelebilmektedir. İlk metot, parametre olarak iletilen tablo nesnesini doldurmak (FillMethodName) amacıyla, diğer metot tablo nesnesini döndürmek (GetMethodName) amacıyla kullanılmaktadır.

Yukarıdaki ekran çıktısında gösterilen özelliklerle üretilen iki metot aşağıda gösterilmiştir.

C#

**public** **partial** **class** KisiTableAdapter : Component {

**public** **virtual** **int** DoldurKisiIdIle(Veriler.KisiDataTable dataTable, **int** Id)  
 { }

**public** **virtual** Veriler.KisiDataTable GetirKisiIdIle(**int** Id)   
 {}

}

VB.NET

**Partial** **Public** **Class** OkuKisiTableAdapter

**Inherits** System.ComponentModel.Component

**Public** **Overloads** **Overridable** **Function** DoldurIdIle(\_  
 **ByVal** dataTable **As** Veriler.OkuKisiDataTable, \_  
 **ByVal** Id **As** **Integer**) **As** **Integer**

**End** **Function**

**Public** **Overloads** **Overridable** **Function** GetirIdIle(**ByVal** Id **As** **Integer**) \_  
 **As** Veriler.OkuKisiDataTable

**End** **Function**

**End** **Class**

Kod 391, Tablo Adaptör Metotları

Üretilen metot isimlerinin VS.NET özellikler penceresindeki değerle aynı olduğuna dikkat ediniz. Üretilen metotlar sanal metotlar olduğu için, daha önce aktarılan sınıf genişletme yöntemlerinden olan miras alma ile bu metotların örtülmesi mümkündür.

Sorgu özellikleri ve açıklamaları aşağıda gösterilmiştir.

| Üye Adı | Açıklama |
| --- | --- |
| CommandText | Komut değerini belirtir. |
| CommandType | Komut tipini belirtir. |
| ExecuteMode | Çalışma tipini belirtir. |
| FillMethodModifier | Tablo doldurma amacıyla üretilen metot giriş niteleyicisini belirtir. |
| FillMethodName | Tablo doldurma amacıyla üretilen metot adını belirtir. |
| GenerateMethods | Hangi metotların üretileceğini belirtir. |
| GetMethodModifier | Tablo döndürme amacıyla üretilen metot giriş niteleyicisini belirtir. |
| GetMethodName | Tablo döndürme amacıyla üretilen metot adını belirtir. |
| Parameters | Sorgu parametrelerini belirtir. |

Tablo 155, Tablo Adaptör Sorgusu Özellikleri

Parametrelerinin Düzenlenmesi

CommandText özelliği ile belirtilen komut içerisinde bulunan parametreler görsel olarak düzenlenebilmektedir. Aşağıda parametrelerin düzenlenmesini sağlayan parametre editör penceresi gösterilmiştir.

Ekran Çıktısı 117, Komut Parametrelerinin Düzenlenmesi

Parametre editörü yardımıyla yeni parametreler eklenmesi ve mevcut parametre özelliklerinin değiştirilmesi mümkündür.

Parametreler, tablo adaptör sınıfı içerisinde kullanılan komut nesneleri ve üretilen metot parametrelerine karşılık gelmektedir. Örneğin parametrenin AllowDbNull özelliğine True değerinin atanmasıyla değersizleşebilen (ing. Nullable) parametre tipleri kaynak kod içerisinde oluşturulmaktadır.

Yeni Sorgular Eklenmesi

Tablo adaptörüne yeni metotlar eklemek için VS.NET veri kümesi editörü içerisinde tablo adaptörünün üzerinde Add Query … komutu verilmelidir.

Ekran Çıktısı 118, Tablo Adaptörüne Yeni Sorgular Eklenmesi

Komutun verilmesiyle önceki konu başlıklarında aktarılan sihirbaz çalışmakta ve sihirbaz yardımıyla yeni sorgular eklenebilmektedir.

Tekil Değer ve Sonuç Döndürmeyen Komutlar

ExecuteMode özelliği, sorgu ile yapılan işlem sonucunda dönmesi beklenen bilgi tipini ifade etmektedir.

Ekran Çıktısı 119, Sorgu Çalışma Tipi

Reader, sorgunun sonuç döndüren tipte olduğunu, Scalar tekil değerler ve NonQuery sonuç döndürmeyen SQL cümlesini belirtmektedir.

Örneğin GetirKisiSayisi isimli depolanmış prosedür, parametre olarak iletilen ad ve soyadı kriterlerine uyan kayıt sayısını döndürmektedir. Bu prosedür ile amaçlanan tekil değerin veritabanından alınmasıdır.

Tablo adaptörüne yeni bir sorgu eklenerek çalışma tipi belirtilebilir.

Ekran Çıktısı 120, Tekil Değer Döndüren Depolanmış Prosedür

Bir sonraki adımda çalışma tipi seçenekleri görüntülenmektedir.

Ekran Çıktısı 121, Sorgu Çalışma Tipinin Belirtilmesi

Son adım, seçilen prosedürün çağrılması için tablo adaptör sınıfında üretilmesi istenen metot adını belirtmektedir.

Ekran Çıktısı 122, Tekil Metot İsminin Belirtilmesi

İşlemin tamamlanması ile oluşturulan sorgu ve özellikleri aşağıda gösterilmiştir.

Ekran Çıktısı 123, Tekil Değer Sorgu Özellikleri

Çalışma tipinin Scalar olarak belirtilmesi ile tablo adaptörü içerisinde oluşturulan metot dönüş değeri de değişmektedir.

C#

**public** **partial** **class** KisiTableAdapter : Component {

**public** **virtual** **object** GetirKisiSayisiAdSoyadIle(**string** Ad, **string** Soyad)

{}

}

VB.NET

**Partial** **Public** **Class** OkuKisiTableAdapter

**Inherits** System.ComponentModel.Component

**Public** **Overridable** **Function** GetirKisiSayisiAdSoyadIle(**ByVal** Ad **As** **String**, \_  
 **ByVal** Soyad **As** **String**) **As** **Object**

**End** **Function**

**End** **Class**

Kod 392, Tekil Değer Metotları

Veri İlişkileri Kullanımı

Veri ilişikleri (ing. Data Relations), veri kümesi içerisinde bulunan tablolar arasındaki ana / detay ilişkisini belirtmek için kullanılmaktadır.

Veri İlişkileri Ekleme

Veri ilişki nesnesi eklemek için VS.NET veri kümesi üzerinde sağ tıklanarak Add | Relation komutu verilmelidir.

Ekran Çıktısı 124, Veri İlişki Nesnesi Ekleme

Komutun verilmesiyle, ana ve detay tablo isimleri ile bu tabloların hangi alanlar üzerinden birbirlerine bağlandıklarının belirtilebileceği editör açılmaktadır.

Ekran Çıktısı 125, Veri İlişki Editörü

İşlemin tamamlanmasıyla veri ilişki nesnesi oluşturulacaktır.

Ekran Çıktısı 126, Kisi ve Adres Tablosu Ana / Detay İlişkisi

Sınıf Yapısı

Kisi ve Adres tabloları veri ilişki nesnesi ile birbirlerine 1 → N, bir başka anlatımla, kişi tablosundaki bir kayda karşılık Adres tablosunda birden fazla kaydın olabilmesi şeklinde bağlanmıştır. Yapılan bu işlem ile, üretilen adres satır sınıfı (AdresRow) içerisine, adres kaydının bağlı olduğu kişi kaydına erişim amacıyla kullanılabilecek kişi satır sınıfı (KisiRow) tipinde özellik eklenmektedir.

Sınıf Kullanım Örnekleri

VS.NET tarafından oluşturulan veri kümesi, tablosu, adaptör ve ilişki sınıflarının kullanımı hakkında bu konu başlığında çeşitli örnekler sunulacaktır.

Sunulan örneklerde kullanılan veri yapısı aşağıda gösterilmiştir.

Ekran Çıktısı 127, Kisi, Adres ve Sehir Tabloları

Üretilen sınıfların veri kümesi ve ilişkileri olmadan kullanım yöntemleri aşağıda gösterilmiştir.

C#

// Tablo adaptörlerini oluştur.

KisiTableAdapter kisiAdaptor = **new** KisiTableAdapter();

AdresTableAdapter adresAdaptor = **new** AdresTableAdapter();

SehirTableAdapter sehirAdaptor = **new** SehirTableAdapter();

**int** kisiId = 0;

kisiAdaptor.Insert(**ref** kisiId, 5467981235, "Melik", "Kanık",   
 **new** DateTime(1999, 6, 5));

Console.WriteLine("Yeni kişi eklendi, kayıt numarası: {0}", kisiId);

kisiAdaptor.Update(kisiId, 3467981234, "Murat", "Kanık",   
 **new** DateTime(1999, 6, 5));

Console.WriteLine("Eklenen kişi kaydı güncellendi");

Veriler.KisiDataTable kisiler = kisiAdaptor.GetirAdSoyadIle("M", **null**);

Console.WriteLine("Adı M ile başlayan kişiler ...");

**foreach** (Veriler.KisiRow kisirow **in** kisiler.Rows)

Console.WriteLine("{0} {1}", kisirow.Ad, kisirow.Soyad);

Veriler.SehirDataTable sehir = sehirAdaptor.GetirTumSehirler();

// Eklenen kişiye ait adres kayıtlarını oluştur.

**int** adresId = 0;

adresAdaptor.Insert(**ref** adresId, kisiId, 1, "51900", "Ev adresi");

adresAdaptor.Insert(**ref** adresId, kisiId, 2, "51900", "İş adresi");

Veriler.AdresDataTable adresler = adresAdaptor.GetirKisiIle(kisiId);

Veriler.KisiRow kisiRow = kisiler.FindById(adresler[0].KisiId);

Console.WriteLine("TcKimlikNo, Ad, Soyadı: {0}, {1}, {2}",   
 kisiRow.TcKimlikNo, kisiRow.Ad, kisiRow.Soyad);

// Adres bilgilerinde bulunan SehirId değeri ile şehir isimlerine ulaş.

**foreach** (Veriler.AdresRow adresRow **in** adresler.Rows)

Console.WriteLine("Adres, Şehir: {0}, {1}", adresRow.Adres,   
 sehir.FindById(adresRow.SehirId).Ad);

// Oluşturulan iki adres verisini sil.

adresler[0].Delete();

adresler[1].Delete();

// Veri tablosunda yapılan değişiklikleri veritabanına ilet.

**int** etkilenenKayitSayisi = adresAdaptor.Update(adresler);

Console.WriteLine("Adres değişiklikleri verit . . .Etkilenen kayıt: {0}",   
 etkilenenKayitSayisi);

// Id değeri ile yeni eklenen kişiyi veritabanından sil.

kisiAdaptor.Delete(kisiId);

Console.WriteLine("Yeni eklenen kişi kaydı silindi.");

VB.NET

' Tablo adaptörlerini oluştur.

**Dim** kisiAdaptor **As** KisiTableAdapter = **New** KisiTableAdapter()

**Dim** adresAdaptor **As** AdresTableAdapter = **New** AdresTableAdapter()

**Dim** sehirAdaptor **As** SehirTableAdapter = **New** SehirTableAdapter()

**Dim** kisiId **As** **Integer**

kisiAdaptor.Insert(kisiId, 5467981235, "Melik", "Kanık", \_  
 **New** DateTime(1999, 6, 5))

Console.WriteLine("Yeni kişi eklendi, kayıt numarası: {0}", kisiId)

kisiAdaptor.Update(kisiId, 3467981234, "Murat", "Kanık", \_  
 **New** DateTime(1999, 6, 5))

Console.WriteLine("Eklenen kişi kaydı güncellendi")

**Dim** kisiler **As** Veriler.KisiDataTable = kisiAdaptor.GetirAdSoyadIle("M", \_   
 **Nothing**)

Console.WriteLine("Adı M ile başlayan kişiler ...")

**For** Each kisirow **As** Veriler.KisiRow **In** kisiler.Rows

Console.WriteLine("{0} {1}", kisirow.Ad, kisirow.Soyad)

**Next**

**Dim** sehir **As** Veriler.SehirDataTable = sehirAdaptor.GetirTumSehirler()

' Eklenen kişiye ait adres kayıtlarını oluştur.

**Dim** adresId **As** **Integer**

adresAdaptor.Insert(adresId, kisiId, 1, "51900", "Ev adresi")

adresAdaptor.Insert(adresId, kisiId, 2, "51900", "İş adresi")

**Dim** adresler **As** Veriler.AdresDataTable = adresAdaptor.GetirKisiIle(kisiId)

**Dim** kisiRow **As** Veriler.KisiRow = kisiler.FindById(adresler(0).KisiId)

Console.WriteLine("TcKimlikNo, Ad, Soyadı: {0}, {1}, {2}", \_

kisiRow.TcKimlikNo, kisiRow.Ad, kisiRow.Soyad)

' Adres bilgilerinde bulunan SehirId değeri ile şehir isimlerine ulaş.

**For** **Each** adresRow **As** Veriler.AdresRow **In** adresler.Rows

Console.WriteLine("Adres, Şehir: {0}, {1}", adresRow.Adres, \_  
 sehir.FindById(adresRow.SehirId).Ad)

**Next**

' Oluşturulan iki adres verisini sil.

adresler(0).Delete()

adresler(1).Delete()

' Veri tablosunda yapılan değişiklikleri veritabanına ilet.

**Dim** etkilenenKayitSayisi **As** **Integer** = adresAdaptor.Update(adresler)

Console.WriteLine("Adres değişiklikleri veritab. . . Etkilenen kayıt: {0}", \_  
 etkilenenKayitSayisi)

' Id değeri ile yeni eklenen kişiyi veritabanından sil.

kisiAdaptor.Delete(kisiId)

Console.WriteLine("Yeni eklenen kişi kaydı silindi.")

Kod 393, Tablo Adaptörleri ve Veri Tabloları Kullanım Örnekleri

Kod çıktısı aşağıda gösterilmiştir.

Yeni kişi eklendi, kayıt numarası: 12

Eklenen kişi kaydı güncellendi

Adı M ile başlayan kişiler ...

Murat Yılmaz

Mustafa Yılmaz

Murat Kanık

TcKimlikNo, Ad, Soyadı: 3467981234, Murat, Kanık

Adres, Şehir: Ev adresi, ADANA

Adres, Şehir: İş adresi, ADIYAMAN

Adres değişiklikleri veritabanına iletildi.Etkilenen kayıt: 2

Yeni eklenen kişi kaydı silindi.

Kod parçasından da görüleceği üzere, veri tablosu nesneleri bağımsız olarak kullanılabilmektedir ve veri kümesi nesnesinin kullanımı zorunlu değildir. Aşağıdaki kod parçasında aynı işlemin veri kümeleri ve ilişkileri kullanılarak yapılması gösterilmiştir.

C#

KisiTableAdapter kisiAdaptor = **new** KisiTableAdapter();

AdresTableAdapter adresAdaptor = **new** AdresTableAdapter();

SehirTableAdapter sehirAdaptor = **new** SehirTableAdapter();

// Veri kümesi nesnesi oluştur.

Veriler veriKumesi = **new** Veriler();

**int** kisiId = 0;

kisiAdaptor.Insert(**ref** kisiId, 5467981235, "Melik", "Kanık",   
 **new** DateTime(1999, 6, 5));

Console.WriteLine("Yeni kişi eklendi, kayıt numarası: {0}", kisiId);

kisiAdaptor.Update(kisiId, 3467981234, "Murat", "Kanık",   
 **new** DateTime(1999, 6, 5));

Console.WriteLine("Eklenen kişi kaydı güncellendi");

// veri kümesi içerisindeki Kisi isimli tabloyu doldur.

kisiAdaptor.DoldurAdSoyadIle(veriKumesi.Kisi,"M", **null**);

Console.WriteLine("Adı M ile başlayan kişiler ...");

**foreach** (Veriler.KisiRow kisirow **in** veriKumesi.Kisi)

Console.WriteLine("{0} {1}", kisirow.Ad, kisirow.Soyad);

sehirAdaptor.DoldurTumSehirler(veriKumesi.Sehir);

**int** adresId = 0;

adresAdaptor.Insert(**ref** adresId, kisiId, 1, "51900", "Ev adresi");

adresAdaptor.Insert(**ref** adresId, kisiId, 2, "51900", "İş adresi");

adresAdaptor.DoldurKisiIle(veriKumesi.Adres, kisiId);

Console.WriteLine("TcKimlikNo, Ad, Soyadı: {0}, {1}, {2}",  
 veriKumesi.Adres[0].KisiRow.TcKimlikNo,   
 veriKumesi.Adres[0].KisiRow.Ad,   
 veriKumesi.Adres[0].KisiRow.Soyad);

// Veri ilişkisi yardımıyla şehir adına eriş.

**foreach** (Veriler.AdresRow adresRow **in** veriKumesi.Adres.Rows)

Console.WriteLine("Adres, Şehir: {0}, {1}", adresRow.Adres,   
 adresRow.SehirRow.Ad);

veriKumesi.Adres[0].Delete();

veriKumesi.Adres[1].Delete();

**int** etkilenenKayitSayisi = adresAdaptor.Update(veriKumesi.Adres);

Console.WriteLine("Adres değişiklikleri veritab. . .Etkilenen kayıt: {0}",   
 etkilenenKayitSayisi);

kisiAdaptor.Delete(kisiId);

Console.WriteLine("Yeni eklenen kisi kaydı silindi.");

VB.NET

**Dim** kisiAdaptor **As** KisiTableAdapter = **New** KisiTableAdapter()

**Dim** adresAdaptor **As** AdresTableAdapter = **New** AdresTableAdapter()

**Dim** sehirAdaptor **As** SehirTableAdapter = **New** SehirTableAdapter()

' Veri kümesi nesnesi oluştur.

**Dim** veriKumesi **As** Veriler = **New** Veriler()

**Dim** kisiId **As** **Integer**

kisiAdaptor.Insert(kisiId, 5467981235, "Melik", "Kanık", \_  
 **New** DateTime(1999, 6, 5))

Console.WriteLine("Yeni kişi eklendi, kayıt numarası: {0}", kisiId)

kisiAdaptor.Update(kisiId, 3467981234, "Murat", "Kanık", \_  
 **New** DateTime(1999, 6, 5))

Console.WriteLine("Eklenen kişi kaydı güncellendi")

' veri kümesi içerisindeki Kisi isimli tabloyu doldur.

kisiAdaptor.DoldurAdSoyadIle(veriKumesi.Kisi, "M", **Nothing**)

Console.WriteLine("Adı M ile başlayan kişiler ...")

**For** **Each** kisirow **As** Veriler.KisiRow **In** veriKumesi.Kisi

Console.WriteLine("{0} {1}", kisirow.Ad, kisirow.Soyad)

**Next**

sehirAdaptor.DoldurTumSehirler(veriKumesi.Sehir)

**Dim** adresId **As** **Integer**

adresAdaptor.Insert(adresId, kisiId, 1, "51900", "Ev adresi")

adresAdaptor.Insert(adresId, kisiId, 2, "51900", "İş adresi")

adresAdaptor.DoldurKisiIle(veriKumesi.Adres, kisiId)

Console.WriteLine("TcKimlikNo, Ad, Soyadı: {0}, {1}, {2}", \_  
 veriKumesi.Adres(0).KisiRow.TcKimlikNo, \_  
 veriKumesi.Adres(0).KisiRow.Ad, \_  
 veriKumesi.Adres(0).KisiRow.Soyad)

' Veri ilişkisi yardımıyla şehir adına eriş.

**For** **Each** AdresRow **As** Veriler.AdresRow **In** veriKumesi.Adres.Rows

Console.WriteLine("Adres, Şehir: {0}, {1}", AdresRow.Adres, \_  
 AdresRow.SehirRow.Ad)

**Next**

veriKumesi.Adres(0).Delete()

veriKumesi.Adres(1).Delete()

**Dim** etkilenenKayitSayisi **As** **Integer** = adresAdaptor.Update(veriKumesi.Adres)

Console.WriteLine("Adres değişiklikleri veritab . . .Etkilenen kayıt: {0}", \_  
 etkilenenKayitSayisi)

kisiAdaptor.Delete(kisiId)

Console.WriteLine("Yeni eklenen kisi kaydı silindi.")

Kod 394, Veri Kümeleri ve İlişkileri Kullanım Örnekleri

Dehanın 10'da 1'i yetenek 10'da 9'u da çalışmaktır.

Albert Einstein

Bölüm

27

.NET Veri Bağlama Mimarisi

V

eritabanı uygulamaları ara yüz kontrolleri yardımıyla verinin kullanıcıya gösterilmesi, gösterilen verinin kullanıcı tarafından düzenlenmesi ve veritabanına kaydedilmesi temel adımlarından oluşmaktadır.

Bu bölümde, .NET tarafından sunulan veri bağlama mimarisi hakkında bilgiler sunulacaktır.

Temel Kavramlar

Veri bağlama (ing. Data Binding), verinin kullanıcı ara yüz kontrollerine iletilerek gösterilmesi ve düzenlenmesinin belirli yöntem ve kurallarla yönetilmesidir.

ASP.NET ve Windows Forms.NET uygulamalarında veri bağlamanın yönetimi için standartlaştırılmış veri bağlama mimarisi (ing. Data Binding Architecture) .NET tarafından sunulmaktadır. .NET 2.0 sürümünden itibaren farklı kullanıcı ara yüz kontrollerinin aynı veri kaynağına bağlanabilmesi ve ileri düzey tasarım desteği bu mimariye ek olarak sağlanmaktadır.

Model View Controller – MVC tasarım şablonunun (ing. Design Pattern) rahatlıkla uygulanabileceği bu mimarinin büyük bölümü System.ComponentModel ve System.Collections ad alanı içerisinde bulunan arabirimler ile yönetilmektedir.

Veritabanı programlama yapılırken sıklıkla kullanılan veri bağlamanın tüm detaylarıyla biliniyor olması karşılaşılan problemlerin çözümünde önemli rol oynayabilmektedir.

Konuya başlamadan önce kitabın koleksiyonlar bölümünün gözden geçirilmesi önerilmektedir.

Örnek Mimari

Veri bağlama ile ifade edilen kavramın daha iyi anlaşılabilmesini sağlamak amacıyla konunun anlatımında farklı mimarilere sahip projeler kullanılacaktır. Projelerin aktarımında, okuyucunun .NET konusunda önünü açacağı değerlendirilen tasarım yöntemleri ve temel düzey bilgiler de bu konu başlığında sunulacaktır.

Çözüm Mimarisi

Çözüm mimarisi aşağıda şematik olarak aşağıda gösterilmiştir.

Şekil 58, Veri Bağlama Örnek Çözüm Mimarisi

* Model projesi, diğer projeler tarafından ortak kullanılan ve sınıf kütüphanesi tipinde olan projedir.
* WindowsApp projesi Windows Forms.NET uygulamasıdır. Bu uygulama içerisinde bulunan Windows pencereleri ve kontrolleri yardımıyla kullanıcılar verileri görüntülemekte ve düzenlemektedir.
* WebApp projesi ASP.NET web uygulaması projesidir. Web uygulamasına internet tarayıcıları ile erişen kullanıcılar HTML kontrolleri yardımıyla verileri görüntülemekte ve düzenlemektedir.
* WebService projesi ASP.NET web hizmet uygulaması projesidir. Uygulama SOAP kullanarak istekleri almakta ve sonuçları iletmektedir.
* PdaApp projesi Windows CE işletim sisteminin çalıştığı akıllı aygıtların (ing. Smart Device) WebService uygulaması tarafından sunulan XML web hizmetlerine erişebilmesini sağlamaktadır. Uygulamanın çalışabilmesi için Microsoft.NET Compact Framework yazılımının aygıt üzerinde kurulmuş olması gerekmektedir.

Mimarinin uygulanması için aşağıdaki adımlar izlenmelidir.

Ekran çıktıları C# projeleri kullanılarak alınmıştır. VB.NET projeleri için aynı yapı ve komutlar geçerlidir.

VS.NET Çözümünün Oluşturulması

Boş bir çözüm oluşturmak için File | New | Project … komutu veriniz.

Ekran Çıktısı 128, VS.NET Çözümü Oluşturulması

Model Projesi

Model projesini eklemek için çözüm üzerinde sağ tıklayarak Add | New Project … komutu veriniz.

Ekran Çıktısı 129, Model Projesi Eklenmesi

Model projesi uygulama iş ve veri katmanını oluşturmaktadır. Bu katman içerisinde bulunan sınıflar yardımıyla farklı mimarilere sahip olan diğer projeler veriye erişmektedirler.

Aşağıdaki ekran çıktısında projeye veri kümesi eklenmesi gösterilmiştir.

Ekran Çıktısı 130, Klasör Açılması ve Projeye Yeni Eleman Eklenmesi

Kitabın önceki bölümlerinde aktarıldığı şekliyle Kisi tablosunun veri kümesi içerisine eklenmesi ve özelliklerinin belirlenmesi ile elde edilen ekran çıktısı aşağıda gösterilmiştir.

Ekran Çıktısı 131, Veri Tablosu, Sorgu ve Çözüm Penceresi

Tablo adaptörleri yardımıyla farklı katmanlardan veriye erişim sağlanabilmektedir.

WindowsApp Projesi

WindowsApp projesini eklemek için çözüm üzerinde sağ tıklayarak Add | New Project … komutu veriniz.

Ekran Çıktısı 132, WindowsApp Projesi Eklenmesi

İşlemin tamamlanmasıyla içerisinde Form1 isminde Windows formunun bulunduğu proje çözüme eklenmektedir.

WebApp Projesi

WebApp projesini eklemek için çözüm üzerinde sağ tıklayarak Add | New Project … komutu veriniz.

Ekran Çıktısı 133, WebApp Projesi Eklenmesi

ASP.NET web uygulama ve hizmetlerinin çalıştırılabilmesi için web sunucusunun (ing. Web Server) sistem üzerinde bulunması gerekmektedir. Web sunucusu olarak VS.NET ile birlikte kurulumu yapılan ve yazılım geliştirilen bilgisayar için hazırlanmış olan Visual Studio Development Server yazılımı kullanılabilmektedir.

Proje özellikleri Web sekmesinde hangi sunucunun kullanılmak istendiği belirtilebilmektedir.

Ekran Çıktısı 134, Web Projesi Özellikler Penceresi

Ekran çıktısında gösterilen ayarlamalar, web sunucusunun 8080 numaralı port üzerinde çalışmasını sağlamaktadır. Sanal yol (ing. Virtual Path) olarak WebApp değerinin belirtilmesiyle http://localhost:8080/WebApp adresi uygulamaya erişim için kullanılmaktadır.

Geliştirme ortamında VS.NET geliştirme sunucusu kullanılabilmesine rağmen üretim ortamlarında IIS (Internet Information Server) kullanılmaktadır. Geliştirme ortamında web sunucusu olarak IIS tercih edilebilir. Bunun için IIS yazılımının kurulmuş ve çalışıyor olması gerekmektedir.

Ekran Çıktısı 135, Geliştirme Ortamında IIS Kullanımı

Ekran çıktısında gösterilen ayarlamalar yapıldıktan sonra Create Virtual Directory komutunun verilmesi ile sanal klasör oluşturulmaktadır. IIS ayarlarının ön tanımlı web port olan 80 değerini içermesi durumunda http://localhost/WebApp adresi uygulamaya erişim için kullanılabilecektir.

IIS üzerinde sanal klasör oluşturmanın bir diğer yolu da dosya sisteminde klasör üzerinde sağ tıklanarak Sharing and Security … komutu verilerek Web Sharing sekmesinin kullanılmasıdır.

Projenin çözüme eklenmesiyle default.aspx isminde web sayfası proje içerisinde oluşturulmaktadır. Bu sayfa üzerinde sağ tıklanarak verilen View in Browser komutu ile sayfanın görüntülenmesi sağlanmaktadır.

WebService Projesi

WebService projesini eklemek için çözüm üzerinde sağ tıklayarak Add | New Project … komutu veriniz.

Ekran Çıktısı 136, WebService Projesi Eklenmesi

Web hizmet projeleri, web uygulama projeleri gibi ASP.NET mimarisini kullanmaktadır. Bu uygulama için de aşağıdaki ekran çıktısında gösterilen ayarlamaların yapılması önerilmektedir.

Ekran Çıktısı 137, Web Hizmet Uygulaması Özellikleri

Ayarlamaların yapılmasıyla http://localhost:8081/WebService adresi uygulamaya erişim için kullanılabilecektir.

Projeye otomatik olarak eklenen service1.asmx isimli dosya web hizmetini simgelemektedir. Bu dosyanın adının servis.asmx olarak değiştirilmesi önerilmektedir. Bir web hizmet uygulaması içerisinde birden çok web hizmeti bulunabilmekte ve uzantısı .asmx olan dosyalar web hizmetine karşılık gelmektedir. Servis.asmx dosyası üzerinde sağ tıklanarak View in Browser komutu verilerek elde edilen ekran çıktısı aşağıda gösterilmiştir.

Ekran Çıktısı 138, Web Hizmetinin İnternet Tarayıcı ile Görüntülenmesi

Adres satırında daha önce proje özellikleri ile yapılan ayarlama sonucu oluşan değer bulunduğuna dikkat ediniz.

Service1, Code Behind olarak adlandırılan ve .aspx veya .asmx dosyalarının kaynak kodlarının bulunduğu dosya içerisinde tanımlı sınıf adını belirtmektedir. Yeni bir web hizmet projesi eklenmesiyle HelloWorld isminde örnek web metodu (ing. Web Method) bu sınıf içerisine eklenmektedir.

C#

[WebService(Namespace = "http://tempuri.org/")]

**public** **class** Service1 : System.Web.Services.WebService {

[WebMethod]

**public** **string** HelloWorld() {

**return** "Hello World";

}

}

VB.NET

<System.Web.Services.WebService(Namespace:="http://tempuri.org/")> \_

**Public** **Class** Service1

**Inherits** System.Web.Services.WebService

<WebMethod()> \_

**Public** **Function** HelloWorld() **As** **String**

**Return** "Hello World"

**End** **Function**

**End** **Class**

Kod 395, Ön Tanımlı Web Hizmet Sınıfı

Sınıf WebService özniteliği ile donatılmıştır. Bu öznitelik farklı web hizmet özelliklerinin belirtilmesi amacıyla kullanılmaktadır. Örneğin web hizmetinin hangi ad alanı içerisinde veya hangi standartlarda olduğu bu öznitelik yardımıyla belirtilebilmektedir.

HelloWorld isimli metodun WebMethod özniteliği ile donatılmış olması bu metodun web hizmet metodu olduğunu belirtmektedir.

Aşağıda web hizmet dosyası (.asmx) içeriği gösterilmiştir.

C#

<%@ WebService Language="C#" CodeBehind="Servis.asmx.cs" Class="WebService.Service1" %>

VB.NET

<%@ WebService Language="VB" CodeBehind="Servis.asmx.vb" Class="WebService.Service1" %>

Kod 396, Web Hizmet Dosyası İçeriği

Gerek web uygulamaları, gerek web hizmetlerinde ASP.NET tarafından yapılan işlem .aspx veya .asmx dosyaları tarafından belirtilen kaynak kod dosyasının derlenmesi ve Service1 isimli sınıfın oluşturulmasıdır.

Web hizmet dosyası üzerinde sağ tıklanarak View Designer komutunun verilmesiyle tasarım penceresi açılmaktadır.

Ekran Çıktısı 139, View Designer Komutu

Sınıf adı .asmx ve kaynak kod dosyası içerisinde bulunduğu için, yeniden adlandırmada tasarım ortamının ve özellikler penceresinin kullanımı önerilmektedir.

Ekran Çıktısı 140, Web Hizmet Sınıf Adının Değiştirilmesi

Yapılan düzenlemeler sonucunda sınıf tanımı aşağıda gösterilmiştir.

C#

[WebService(Namespace = "http://ns.ornekdomain.com/")]

**public** **class** Servis : System.Web.Services.WebService {

[WebMethod]

**public** **int** Topla(**int** x, **int** y) {

**return** x + y;

}

}

VB.NET

<System.Web.Services.WebService(**Namespace**:=" http://ns.ornekdomain.com /")> \_

**Public** **Class** Servis

**Inherits** System.Web.Services.WebService

<WebMethod()> \_

**Public** **Function** Topla(**ByVal** x **As** **Integer**, **ByVal** y **As** **Integer**) **As** **Integer**

**Return** x + y

**End** **Function**

**End** **Class**

Kod 397, Web Hizmet Sınıfı ve Düzenlemeler

Projenin derlenerek internet tarayıcında görüntülenmesi sonucu elde edilen ekran çıktısı aşağıda gösterilmiştir.

Ekran Çıktısı 141, Web Hizmet Düzenlemeleri ve Topla Metodu

PdaApp Projesi

PdaApp projesini eklemek için çözüm üzerinde sağ tıklayarak Add | New Project … komutu veriniz.

Ekran Çıktısı 142, PdaApp Projesi Eklenmesi

Projenin eklenmesiyle Form1 isminde akıllı aygıt formu oluşturulmaktadır.

Ekran Çıktısı 143, PdaApp Formu

Akıllı aygıt uygulamalarının hata ayıklama işlemleri ve çalıştırılabilmeleri için cihazın fiziksel olarak bilgisayara bağlı olması veya emülatörler yardımıyla bu işlemlerin yapılması gerekmektedir. Proje çıktılarının cihaza dağıtımı (ing. Deployment) VS.NET tarafından otomatik olarak yapılmaktadır.

Platform bazında yüklü cihazların görüntülenmesi ve çeşitli ayarlarının yapılabilmesi amacıyla Tools | Options komutu ile VS.NET ayarları penceresi açılmalıdır.

Ekran Çıktısı 144, VS.NET Aygıt Ayarları Sekmesi

Proje özellikleri Devices sekmesi yardımıyla projenin çalıştırılmasında kullanılmak istenen cihaz ayarlanmaktadır.

Ekran Çıktısı 145, Windows CE Proje Özellikleri

Proje üzerinde sağ tıklanarak verilen Deploy komutu ile seçilen cihaza yükleme yapılmaktadır. Projenin çalıştırılmasıyla emülatör görüntülenmekte ve uygulama emülatör içerisinde çalışmaktadır.

Aşağıdaki ekran çıktısında Pocket PC 2003 SE isimli emülatörde çalışan uygulama gösterilmiştir.

Ekran Çıktısı 146, Emülatör ve PdaApp Uygulaması

Proje Referanslarının Belirlenmesi

Tasarım mimarisi gereği PdaApp projesi dışındaki tüm projelerin Model projesine referanslarının bulunması gerekmektedir. Bunun için References üzerinde sağ tıklanarak Add Reference … komutunun verilmesi gerekmektedir.

Ekran Çıktısı 147, Proje Referansı Eklenmesi

PdaApp projesi WebService tarafından sağlanan web hizmetlerini kullanmaktadır. Web hizmetine referans vermek için öncelikle VS.NET geliştirme sunucusunun çalışıyor olması gerekmektedir. Bunun için uygulanabilecek en pratik yöntem servis.asmx dosyası üzerinde sağ tıklayarak internet tarayıcısında görüntülenmesidir. VS.NET geliştirme sunucusu Windows Task Bar üzerinde Notification Area içerisinde bulunmaktadır.

Web hizmetine referans vermek için PdaApp References üzerinde sağ tıklanarak Add Web Reference … komutu verilmelidir.

Ekran Çıktısı 148, Web Referansı Eklenmesi

Komutun verilmesiyle Web hizmet referansları eklemek için kullanılan pencere açılacaktır.

Ekran Çıktısı 149, Web Referans Ekleme Penceresi

Add Reference komutunun verilmesiyle PdaApp.WebServiceRef ad alanı içerisinde Servis isimli sınıf otomatik olarak oluşturulmaktadır.

Ekran Çıktısı 150, Proje Web Referansları

Web Metotları Kullanımı

Web hizmet proksi sınıfı (ing. Web Service Proxy Class) olarak adlandırılan Servis sınıfı tarafından yapılan işlem WebService tarafından sunulan web metotlarının çağrılmasıdır.

Aşağıda hizmetin kullanılabilmesi için form üzerinde yapılan tasarım gösterilmiştir.

Ekran Çıktısı 151, Tasarım Anında Akıllı Aygıt Formu

Hesapla komutunun verilmesiyle örnek proksi sınıfı kullanım aşağıda gösterilmiştir.

C#

Servis hizmet = **new** Servis();

hizmet.Url = "http://10.10.1.1/WebService/Servis.asmx";

**int** sayi1 = **int**.Parse(ctlSayi1.Text);

**int** sayi2 = **int**.Parse(ctlSayi2.Text);

**int** sonuc = hizmet.Topla(sayi1, sayi2);

ctlSonuc.Text = **string**.Format("Sonuc: {0}", sonuc);

VB.NET

**Dim** hizmet **As** Servis = **New** Servis()

hizmet.Url = "http://10.10.1.1/WebService/Servis.asmx"

**Dim** sayi1 **As** **Integer** = **Integer**.Parse(ctlSayi1.Text)

**Dim** sayi2 **As** **Integer** = **Integer**.Parse(ctlSayi2.Text)

**Dim** sonuc **As** **Integer** = hizmet.Topla(sayi1, sayi2)

ctlSonuc.Text = **String**.Format("Sonuc: {0}", sonuc)

Kod 398, Web Hizmet Proksi Sınıfı Kullanımı

Hizmet nesnesi tarafından kullanılması istenen adresin Url özelliği ile atandığına dikkat edilmelidir. localhost değeri kodun çalıştığı bilgisayar adresini ifade etmektedir. Ön tanımlı değerin (http://localhost:8081/WebService/Servis.asmx) kullanılması durumunda istek direkt emülatöre yapılmış olacaktır.

Ekran Çıktısı 152, Akıllı Aygıt Uygulaması ile Web Hizmetinin Kullanımı

Uygulamanın .NET Compact Framework üzerinde çalıştığına dikkat edilmelidir.

Yapılan işlem, 10.10.1.1 IP adresinde çalışan ASP.NET web hizmet uygulamasının proksi sınıfı ile çağrılması ve sonucun kullanıcıya gösterilmesidir. İşlem detayları şematik olarak aşağıda gösterilmiştir.

Şekil 59, Web Hizmet Metodunun Çağrılması

Web Metotları Gerçekleştirimi

Aşağıdaki kod parçasında kişi bilgisinin web metotları yardımıyla uzaktan düzenlenebilmesi ve seçilebilmesi amacıyla hazırlanan metotlar gösterilmiştir. Metotlar Model içerisinde bulunan tablo adaptörünü kullanmaktadır.

C#

**public** **class** Servis : System.Web.Services.WebService {

[WebMethod]

**public** KisiDs.KisiDataTable GetirAdSoyadIle(**string** ad, **string** soyad) {

KisiTableAdapter adapter = **new** KisiTableAdapter();

**return** adapter.GetirAdSoyadIle(ad, soyad);

}

[WebMethod]

**public** **int** EkleKisi(**ref** **int** Id, **long** TcKimlikNo, **string** Ad, **string** Soyad,   
 DateTime DogumTarih) {

KisiTableAdapter adapter = **new** KisiTableAdapter();

**return** adapter.Insert(**ref** Id, TcKimlikNo, Ad, Soyad, DogumTarih);

}

[WebMethod]

**public** **int** GuncelleKisi(**int** Id, **long** TcKimlikNo, **string** Ad, **string** Soyad,   
 DateTime DogumTarih) {

KisiTableAdapter adapter = **new** KisiTableAdapter();

**return** adapter.Update(Id, TcKimlikNo, Ad, Soyad, DogumTarih);

}

[WebMethod]

**public** **int** SilKisi(**int** Id) {

KisiTableAdapter adapter = **new** KisiTableAdapter();

**return** adapter.Delete(Id);

}

[WebMethod]

**public** **int** GuncelleKisiTablosu(KisiDs.KisiDataTable tablo) {

KisiTableAdapter adapter = **new** KisiTableAdapter();

**return** adapter.Update(tablo);

}

}

VB.NET

**Public** **Class** Servis

**Inherits** System.Web.Services.WebService

<WebMethod()> \_

**Public** **Function** GetirAdSoyadIle(**ByVal** ad **As** **String**, **ByVal** soyad **As** **String**)\_  
 **As** KisiDs.KisiDataTable

**Dim** adapter **As** KisiTableAdapter = **New** KisiTableAdapter()

**Return** adapter.GetirAdSoyadIle(ad, soyad)

**End** **Function**

<WebMethod()> \_

**Public** **Function** EkleKisi(**ByRef** Id **As** **Integer**, **ByVal** TcKimlikNo **As** **Long**, \_  
 **ByVal** Ad **As** **String**, **ByVal** Soyad **As** **String**, \_   
 **ByVal** DogumTarih **As** DateTime) **As** **Integer**

**Dim** adapter **As** KisiTableAdapter = **New** KisiTableAdapter()

**Return** adapter.Insert(Id, TcKimlikNo, Ad, Soyad, DogumTarih)

**End** **Function**

<WebMethod()> \_

**Public** **Function** GuncelleKisi(**ByVal** Id **As** **Integer**, **ByVal** TcKimlikNo **As** **Long**,   
 **ByVal** Ad **As** **String**, **ByVal** Soyad **As** **String**, \_  
 **ByVal** DogumTarih **As** DateTime) **As** **Integer**

**Dim** adapter **As** KisiTableAdapter = **New** KisiTableAdapter()

**Return** adapter.Update(Id, TcKimlikNo, Ad, Soyad, DogumTarih)

**End** **Function**

<WebMethod()> \_

**Public** **Function** SilKisi(**ByVal** Id **As** **Integer**) **As** **Integer**

**Dim** adapter **As** KisiTableAdapter = **New** KisiTableAdapter()

**Return** adapter.Delete(Id)

**End** **Function**

<WebMethod()> \_

**Public** **Function** GuncelleKisiTablosu(**ByVal** tablo **As** KisiDs.KisiDataTable) \_  
 **As** **Integer**

**Dim** adapter **As** KisiTableAdapter = **New** KisiTableAdapter()

**Return** adapter.Update(tablo)

**End** **Function**

**End** **Class**

Kod 399, Web Hizmet Metot Gerçekleştirimleri

İnternet tarayıcısı çıktısı aşağıda gösterilmiştir.

Ekran Çıktısı 153, Web Hizmet Metotları ve İnternet Tarayıcısı

Web hizmetinin yerel sunucu üzerinde görüntülenmesi durumunda internet tarayıcı ile metotların çağrılabilmesi mümkündür.

Örneğin AraKisiAdSoyadIle isimli web hizmet metoduna tıklandığında parametrelerin girilebileceği sayfa açılmaktadır.

Ekran Çıktısı 154, Web Metot Parametrelerinin Belirtilmesi

Invoke komutunun verilmesiyle tüm kişi bilgileri elde edilmektedir.

Ekran Çıktısı 155, Web Hizmet Metot Sonucu

Veri Bağlama Örneği

Veri bağlama, .NET veri bağlama mimarisi tarafından belirlenen kurallara uyularak geliştirilen sınıfların kullanıcı ara yüz kontrolleri tarafından gösterimi ve düzenlenebilmesini ifade etmektedir.

Aktarılan mimari kapsamında örnek veri bağlamanın yapılabilmesi için Windows formuna liste kutusu ve iki adet buton kontrolü eklenmiştir.

Ekran Çıktısı 156, Windows Kontrollerinin Eklenmesi

Benzer işlem ASP.NET web sayfası ve Windows CE formu için aşağıda gösterilmiştir.

Ekran Çıktısı 157, ASP.NET Kontrollerinin Eklenmesi

Ekran Çıktısı 158, Windows CE Kontrollerinin Eklenmesi

Liste kutularında kişi isimlerinin gösterimi hedeflenmektedir. Veri bağlama mimarisi kullanılmadan Kayıtları Göster komutu verildiğinde aşağıdaki şekilde gösterim yapılabilir.

C#

// Windows Forms.NET Uygulaması

**private** **void** ctlKayitGosterButton\_Click(**object** sender, EventArgs e) {

KisiTableAdapter adapter = **new** KisiTableAdapter();

KisiDs.KisiDataTable tablo = adapter.GetirAdSoyadIle(**null**, **null**);

ctlListe.Items.Clear();

**foreach** (KisiDs.KisiRow row **in** tablo.Rows)

ctlListe.Items.Add(row.Ad);

}

// ASP.NET Web Uygulaması

**protected** **void** ctlKayitGosterButton\_Click(**object** sender, EventArgs e)

{

KisiTableAdapter adapter = **new** KisiTableAdapter();

KisiDs.KisiDataTable tablo = adapter.GetirAdSoyadIle(**null**, **null**);

ctlListe.Items.Clear();

**foreach** (KisiDs.KisiRow row **in** tablo.Rows)

ctlListe.Items.Add(row.Ad);

}

// Windows CE Uygulaması

**private** **void** ctlKayitGosterButton\_Click(**object** sender, EventArgs e)

{

Servis hizmet = **new** Servis();

KisiDs.KisiDataTable tablo = hizmet.GetirAdSoyadIle(**null**, **null**);

ctlListe.Items.Clear();

**foreach** (KisiDs.KisiRow row **in** tablo.Rows)

ctlListe.Items.Add(row.Ad);

}

VB.NET

' Windows Forms.NET Uygulaması

**Private** **Sub** ctlKayitGosterButton\_Click(**ByVal** sender **As** System.**Object**, \_  
 **ByVal** e **As** System.EventArgs) \_  
 **Handles** ctlKayitGosterButton.Click

**Dim** adapter **As** KisiTableAdapter = **New** KisiTableAdapter()

**Dim** tablo **As** KisiDs.KisiDataTable = adapter.GetirAdSoyadIle(**Nothing**, \_  
 **Nothing**)

ctlListe.Items.Clear()

**For** **Each** row **As** KisiDs.KisiRow **In** tablo.Rows

ctlListe.Items.Add(row.Ad)

**Next**

**End** **Sub**

' ASP.NET Web Uygulaması

**Protected** **Sub** ctlKayitGosterButton\_Click(**ByVal** sender **As** System.**Object**, \_  
 **ByVal** e **As** System.EventArgs) \_  
 **Handles** ctlKayitGosterButton.Click

**Dim** adapter **As** KisiTableAdapter = **New** KisiTableAdapter()

**Dim** tablo **As** KisiDs.KisiDataTable = adapter.GetirAdSoyadIle(**Nothing**, \_  
 **Nothing**)

ctlListe.Items.Clear()

**For** **Each** row **As** KisiDs.KisiRow **In** tablo.Rows

ctlListe.Items.Add(row.Ad)

**Next**

**End** **Sub**

' Windows CE Uygulaması

**Private** **Sub** ctlKayitGosterButton\_Click(**ByVal** sender **As** System.**Object**, \_  
 **ByVal** e **As** System.EventArgs) \_  
 **Handles** ctlKayitGosterButton.Click

**Dim** hizmet **As** Servis = **New** Servis()

**Dim** tablo **As** KisiDs.KisiDataTable = hizmet.GetirAdSoyadIle(**Nothing**, \_  
 **Nothing**)

ctlListe.Items.Clear()

**For** **Each** row **As** KisiDs.KisiRow **In** tablo.Rows

ctlListe.Items.Add(row.Ad)

**Next**

**End** **Sub**

Kod 400, Klasik Veri Gösterimi

PdaApp projesi web hizmet referansının güncellenmesi için referans üzerinde Update Web Reference komutu verilmelidir. Böylece, web hizmetinde yapılan değişiklikler proksi sınıfına yansıtılmaktadır.

Kod parçasından da görüleceği üzere, tablo içerisindeki her kayıt listeye bir döngü içerisinde eklenmektedir.

Veri bağlama mimarisi daha farklı bir yöntem sunmaktadır.

C#

// Windows Forms.NET Uygulaması

**private** **void** ctlVeriBaglaButton\_Click(**object** sender, EventArgs e) {

KisiTableAdapter adapter = **new** KisiTableAdapter();

KisiDs.KisiDataTable tablo = adapter.GetirAdSoyadIle(**null**, **null**);

ctlListe.DisplayMember = "Ad";

ctlListe.DataSource = tablo;

}

// ASP.NET Web Uygulaması

**protected** **void** ctlVeriBaglaButton\_Click(**object** sender, EventArgs e)

{

KisiTableAdapter adapter = **new** KisiTableAdapter();

KisiDs.KisiDataTable tablo = adapter.GetirAdSoyadIle(**null**, **null**);

ctlListe.DataTextField = "Ad";

ctlListe.DataSource = tablo;

ctlListe.DataBind();

}

// Windows CE Uygulaması

**private** **void** ctlVeriBaglaButton\_Click(**object** sender, EventArgs e)

{

Servis hizmet = **new** Servis();

KisiDs.KisiDataTable tablo = hizmet.GetirAdSoyadIle(**null**, **null**);

ctlListe.DisplayMember = "Ad";

ctlListe.DataSource = tablo;

}

VB.NET

' Windows Forms.NET Uygulaması

**Private** **Sub** ctlVeriBaglaButton\_Click(**ByVal** sender **As** System.**Object**, \_  
 **ByVal** e **As** System.EventArgs)   
 **Handles** ctlVeriBaglaButton.Click

**Dim** adapter **As** KisiTableAdapter = **New** KisiTableAdapter()

**Dim** tablo **As** KisiDs.KisiDataTable = adapter.GetirAdSoyadIle(**Nothing**, \_  
 **Nothing**)

ctlListe.DisplayMember = "Ad"

ctlListe.DataSource = tablo

**End** **Sub**

' ASP.NET Web Uygulaması

**Protected** **Sub** ctlVeriBaglaButton\_Click(**ByVal** sender **As** System.**Object**, \_  
 **ByVal** e **As** System.EventArgs)  
 **Handles** ctlVeriBaglaButton.Click

**Dim** adapter **As** KisiTableAdapter = **New** KisiTableAdapter()

**Dim** tablo **As** KisiDs.KisiDataTable = adapter.GetirAdSoyadIle(**Nothing**, \_  
 **Nothing**)

ctlListe.DataTextField = "Ad"

ctlListe.DataSource = tablo

ctlListe.DataBind()

**End** **Sub**

' Windows CE Uygulaması

**Private** **Sub** ctlVeriBaglaButton\_Click(**ByVal** sender **As** System.**Object**, \_  
 **ByVal** e **As** System.EventArgs) \_  
 **Handles** ctlVeriBaglaButton.Click

**Dim** hizmet **As** Servis = **New** Servis()

**Dim** tablo **As** KisiDs.KisiDataTable = hizmet.GetirAdSoyadIle(**Nothing**, \_  
 **Nothing**)

ctlListe.DisplayMember = "Ad"

ctlListe.DataSource = tablo

**End** **Sub**

Kod 401, Basit Veri Bağlama Örneği

Aşağıdaki ekran çıktılarında veri bağlama sonucu elde edilen liste kutuları gösterilmiştir.

Ekran Çıktısı 159, Windows Uygulaması Veri Bağlama Sonucu

Ekran Çıktısı 160, Web Uygulaması Veri Bağlama Sonucu

Ekran Çıktısı 161, Windows CE Uygulaması Veri Bağlama Sonucu

Mimari, kullanıcı ara yüz kontrollerine bağlanan nesnelerin gerçekleştirimlerini yapmış oldukları arabirimler yardımıyla veriye erişebilmeyi temel almaktadır.

Liste kutusunun veriye erişimi aşağıdaki adımlardan oluşmaktadır.

1. Veri bağlama mimarisi kuralları uyarınca DataSource özelliğine atanan nesne sorgulanmaktadır.
2. DisplayMember veya DataTextField özellikleri ile ifade edilen değerler yansıma (ing. Reflection) yardımıyla elde edilmektedir.
3. Elde edilen değerler liste kutusuna eklenmektedir.

.NET veri bağlama mimarisi hakkında en sık yanılgıya düşülebilen konulardan birisi de veri bağlamanın sadece veritabanı veya veri kümeleri / tabloları ile yapılabildiğidir. Veri bağlama mimarisi tamamen bağımsız olarak çalışmakta ve herhangi bir sınıf veya veritabanı bağımlılığı bulunmamaktadır.

Veri bağlamanın bağımsız şekilde yapılabilmesi ile, belirlenen standartlara uyulduğu sürece, her nesnenin kolaylıkla kullanıcı ara yüz kontrollerinde gösterimi veya düzenlenmesi mümkün olmaktadır. Buna rağmen bağımsız mimari çeşitli zorlukları ve karmaşık bir yapıyı da beraberinde getirmektedir.

Veri Bağlama Mimarisi

.NET veri bağlama mimarisi aşağıda listelenen temel işlevleri destekleyecek şekilde tasarlanmıştır.

1. Verinin kullanıcı ara yüz kontrollerinde gösterimi.
2. Kullanıcı ara yüz kontrollerinde yapılan değişikliklerin veriye yansıtılması.
3. Filtreleme.
4. Sıralama.

Tek yönlü veri bağlama (ing. One-way Data Binding), verinin ara yüz kontrolleri tarafından sadece gösterimini ifade etmektedir. Çift yönlü veri bağlama (ing. Two-way Data Binding), gösterilen veri üzerinde yapılan değişikliklerin de mimari kurallarına uyularak veriye yansıtılabilmesini ifade etmektedir.

Önceki konu başlığında tek yönlü veri bağlama örneklenmiştir. Liste kutusunda veri sadece gösterilmekte, üzerinde herhangi bir değişiklik yapılmamaktadır. Buna rağmen farklı kullanıcı kontrolleri verinin düzenlenebilmesini de sağlamaktadırlar. Örneğin veri tablosu kontrolleri (Grid) veriyi gösterebilmekte ve üzerinde değişiklik yapılabilmesini de desteklemektedir.

Veri Bağlama Arabirimleri

Veri bağlama mimarisi tarafından kullanılan temel arabirimler ve açıklamaları aşağıda listelenmiştir.

| Arabirim | Açıklama |
| --- | --- |
| IList | Temel liste arabirimidir. Arabirim yardımıyla liste içerisindeki elemanlara indeksleri ile erişim sağlanmaktadır. |
| IBindingList | Sıralama ve veri düzenleme desteği sağlayan arabirimdir. |
| IBindingListView | Sıralama ve filtreleme desteği sağlayan arabirimdir. |
| ITypedList | Veri bağlamada kullanılan nesnenin açık özellikleri dışında kullanılması istenen özellikler için destek sağlayan arabirimdir. |
| ISupportInitialize | Bağlanmak istenen nesnenin bu arabirim gerçekleştirimini yapması durumunda nesnenin işleme başlaması ve tamamlaması için gerekli metotlar çağrılmaktadır. |
| ISupoortInitiliazeNotification | Veri bağlama işlemi için, nesnenin hazırlık işlemlerinin tamamlandığını belirten olay tanımının bulunduğu arabirimdir. |
| IListSource | Nesnenin kendisi yerine, veri bağlama amacıyla kullanılmasını istediği nesneleri döndürebilmesini sağlayan arabirimdir. |

Tablo 156, Veri Bağlama Arabirimleri ve Açıklamaları

Veri Kümesi ve Tablosu Sınıfları

Aşağıda System.Data.DataSet sınıf tanımı gösterilmiştir.

C#

**public** **class** DataSet : MarshalByValueComponent, IListSource,   
 IXmlSerializable, ISupportInitializeNotification,   
 ISupportInitialize, ISerializable

VB.NET

**Public** **Class** DataSet

**Inherits** MarshalByValueComponent

**Implements** IListSource, IXmlSerializable, \_  
 ISupportInitializeNotification, ISupportInitialize, \_  
 ISerializable

Kod 402, System.Data.DataSet Sınıf Tanımı

DataSet sınıfı IListSource arabirim gerçekleştirimini yapmaktadır.

IListSource arabirimi, ara yüz kontrolüne bağlanan nesnenin kendisi yerine, nesne tarafından döndürülen listenin bağlama amacıyla kullanılmasını belirtmektedir. Bu anlamda, ara yüz kontrolüne veri kümesi nesnesinin bağlanması durumunda veri bağlamanın veri kümesi yerine veri kümesi tarafından döndürülen liste ile yapıldığına dikkat edilmelidir.

Veri bağlamada sıklıkla kullanılan System.Data.DataTable sınıf tanımı aşağıda gösterilmiştir.

C#

**public** **class** DataTable : MarshalByValueComponent, IListSource,   
 ISupportInitializeNotification, ISupportInitialize,   
 ISerializable, IXmlSerializable

VB.NET

**Public** **Class** DataTable

**Inherits** MarshalByValueComponent

**Implements** IListSource, ISupportInitializeNotification, \_  
 ISupportInitialize, ISerializable, IXmlSerializable

Kod 403, System.Data.DataTable Sınıf Tanımı

Sınıf tanımından, veri tablosu nesnesinin veri bağlamada kullanılması durumunda, veri tablosu nesnesi yerine nesne tarafından döndürülen listenin veri bağlamada kullanıldığı görülmektedir.

DataSet veya DataTable nesnelerinin kullanıcı ara yüz kontrolleri DataSource özelliğine atanması durumunda, atanan nesnenin IListSource arabirim gerçekleştirimini yapıp yapmadığı .NET veri bağlama mimarisi tarafından kontrol edilmektedir. Nesnenin bu arabirim gerçekleştirimini yapması durumunda veri bağlama amacıyla nesne yerine nesne tarafından döndürülen nesne kullanılmaktadır.

Aşağıda IListSource arabirim tanımı gösterilmiştir.

C#

**public** **interface** IListSource {

IList GetList();

**bool** ContainsListCollection { **get**; }

}

VB.NET

**Public** **Interface** IListSource

**Function** GetList() **As** IList

**ReadOnly** **Property** ContainsListCollection **As** **Boolean**

**End** **Interface**

Kod 404, System.ComponentModel.IListSource Arabirim Tanımı

IListSource.GetList arabirim metodu ile döndürülmesi gereken nesnenin IList arabirim gerçekleştirimini yapmış olması gerektiğine dikkat ediniz.

Aşağıdaki kod parçasında DataTable.GetList metot gerçekleştirimi gösterilmiştir.

C#

IList IListSource.GetList() {

**return** **this**.DefaultView;

}

VB.NET

**Function** IListSource.GetList() **As** IList **Implements** IListSource.GetList

**Return** **Me**.DefaultView

**End** **Function**

Kod 405, DataTable.GetList Metot İçeriği

Veri tablo nesnesi tarafından veri bağlamada kullanılmak üzere DataTable.DefaultView özelliği ile ifade edilen nesne döndürülmektedir. DataTable.DefaultView, System.Data.DataView tipinde olan ve veri görünüm nesnesini simgeleyen özelliktir. Buradan çıkartılması gereken sonuç, veri bağlamanın veri tablosu yerine, veri tablosu nesnesi tarafından döndürülen veri görünüm nesnesi ile yapıldığıdır.

Aşağıda veri görünüm sınıf tanımı gösterilmiştir.

C#

**public** **class** DataView : MarshalByValueComponent, IBindingListView,   
 IBindingList, IList, ICollection, IEnumerable,   
 ITypedList, ISupportInitializeNotification,   
 ISupportInitialize

VB.NET

**Public** **Class** DataView

**Inherits** MarshalByValueComponent

**Implements** IBindingListView, IBindingList, IList, ICollection, \_  
 IEnumerable, ITypedList, ISupportInitializeNotification, \_  
 ISupportInitialize

Kod 406, System.Data.DataView Sınıf Tanımı

Sınıf tanımından da görüleceği üzere, System.Data.DataView sınıfı veri bağlama mimarisi tarafından kullanılan arabirim gerçekleştirimlerini yapmaktadır. Böylece, tamamen bağımsız olarak tasarlanmış olan .NET veri bağlama mimarisi bu arabirimler yardımıyla verinin ara yüz kontrolüne nasıl bağlanacağını ve veri düzenlemelerinin nasıl yöneteceğini belirlemektedir.

Veri Bağlama Sınıf Örneği

Aşağıda programcı tanımlı Kisi sınıfı gösterilmiştir.

C#

[Serializable]

**public** **class** KisiBilgisi {

**private** **int** id;

**private** **long** tcKimlikNo;

**private** **string** ad;

**private** **string** soyad;

**private** DateTime dogumTarih;

**public** **int** Id { **get** { **return** id; } **set** { id = value; } }

**public** **long** TcKimlikNo {

**get** {

**return** tcKimlikNo;

}

**set** {

tcKimlikNo = value;

}

}

**public** **string** Ad { **get** { **return** ad; } **set** { ad = value; } }

**public** **string** Soyad { **get** { **return** soyad; } **set** { soyad = value; } }

**public** DateTime DogumTarih {

**get** {

**return** dogumTarih;

}

**set** {

dogumTarih = value;

}

}

**public** KisiBilgisi(**int** id, **long** tcKimlikNo, **string** ad, **string** soyad,   
 DateTime dogumTarih) {

**this**.id = id;

**this**.tcKimlikNo = tcKimlikNo;

**this**.ad = ad;

**this**.soyad = soyad;

**this**.dogumTarih = dogumTarih;

}

}

VB.NET

<Serializable()> \_

**Public** **Class** KisiBilgisi

**Private** \_id **As** **Integer**

**Private** \_tcKimlikNo **As** **Long**

**Private** \_ad **As** **String**

**Private** \_soyad **As** **String**

**Private** \_dogumTarih **As** DateTime

**Public** **Property** Id() **As** **Integer**

**Get**

**Return** \_id

**End** **Get**

**Set**(**ByVal** value **As** **Integer**)

\_id = Value

**End** **Set**

**End** **Property**

**Public** **Property** TcKimlikNo() **As** **Long**

**Get**

**Return** \_tcKimlikNo

**End** **Get**

**Set**(**ByVal** value **As** **Long**)

\_tcKimlikNo = Value

**End** **Set**

**End** **Property**

**Public** **Property** Ad() **As** **String**

**Get**

**Return** \_ad

**End** **Get**

**Set**(**ByVal** value **As** **String**)

\_ad = Value

**End** **Set**

**End** **Property**

**Public** **Property** Soyad() **As** **String**

**Get**

**Return** \_soyad

**End** **Get**

**Set**(**ByVal** value **As** **String**)

\_soyad = Value

**End** **Set**

**End** **Property**

**Public** **Property** DogumTarih() **As** DateTime

**Get**

**Return** \_dogumTarih

**End** **Get**

**Set**(**ByVal** value **As** DateTime)

\_dogumTarih = Value

**End** **Set**

**End** **Property**

**Public** **Sub** **New**(**ByVal** id **As** **Integer**, **ByVal** tcKimlikNo **As** **Long**, \_  
 **ByVal** ad **As** **String**, **ByVal** soyad **As** **String**, \_  
 **ByVal** dogumTarih **As** DateTime)

**Me**.\_id = id

**Me**.\_tcKimlikNo = tcKimlikNo

**Me**.\_ad = ad

**Me**.\_soyad = soyad

**Me**.\_dogumTarih = dogumTarih

**End** **Sub**

**End** **Class**

Kod 407, Programcı Tanımlı Kişi Sınıfı

Kisi nesneleri içeren KisiListesi sınıfı aşağıda gösterilmiştir. KisiListesi, System.Collections.Generic.List sınıfını miras almaktadır.

C#

[Serializable]

**public** **class** KisiListesi: List<KisiBilgisi> {

}

VB.NET

<Serializable()> \_

**Public** **Class** KisiListesi

**Inherits** List(**Of** KisiBilgisi)

**End** **Class**

Kod 408, Programcı Tanımlı KisiListesi Sınıfı

Kişi bilgilerinin yönetiminin yapılabilmesi ve XML olarak kaydedilebilmesi için kullanılan KisiListesiAdaptoru isimli sınıf aşağıda gösterilmiştir.

C#

**public** **class** KisiListesiAdaptoru {

**private** **static** KisiListesi liste = **null**;

**public** **static** KisiListesi GetirListe() {

**if** (liste == **null**)

liste = **new** KisiListesi();

liste.Add(**new** KisiBilgisi(1, 56443344551, "Mehmet", "Yılmaz", \_  
 **new** DateTime(1955, 9, 9)));

liste.Add(**new** KisiBilgisi(2, 36423140671, "Mahmut", "Önder", \_  
 **new** DateTime(1945, 9, 9)));

**return** liste;

}

**public** **static** **void** Kaydet(KisiListesi liste)

{

XmlSerializer serializer = **new** XmlSerializer(**typeof**(KisiListesi));

**using** (FileStream f = **new** FileStream(@"c:\veriler.xml", FileMode.Create)) {

serializer.Serialize(f, liste);

f.Close();

}

}

**public** **static** KisiListesi Yukle() {

**if** (!File.Exists(@"c:\veriler.xml"))

**return** GetirListe();

XmlSerializer serializer = **new** XmlSerializer(**typeof**(KisiListesi));

**object** o;

**using** (FileStream f = **new** FileStream(@"c:\veriler.xml", FileMode.Open)) {

o = serializer.Deserialize(f);

f.Close();

}

**return** (KisiListesi)o;

}

}

VB.NET

**Public** **Class** KisiListesiAdaptoru

**Private** **Shared** liste **As** KisiListesi = **Nothing**

**Public** **Shared** **Function** GetirListe() **As** KisiListesi

**If** liste **Is** **Nothing** **Then**

liste = **New** KisiListesi

**End** **If**

liste.Add(**New** KisiBilgisi(1, 56443344551, "Mehmet", "Yılmaz", \_  
 **New** DateTime(1955, 9, 9)))

liste.Add(**New** KisiBilgisi(2, 36423140671, "Mahmut", "Önder", \_  
 **New** DateTime(1945, 9, 9)))

**Return** liste

**End** **Function**

**Public** **Shared** **Sub** Kaydet(**ByVal** liste **As** KisiListesi)

**Dim** serializer **As** XmlSerializer = \_  
 **New** XmlSerializer(**GetType**(KisiListesi))

**Dim** f **As** FileStream = **New** FileStream("c:\veriler.xml", FileMode.Create)

serializer.Serialize(f, liste)

f.Close()

**End** **Sub**

**Public** **Shared** **Function** Yukle() **As** KisiListesi

**If** **Not** File.Exists("c:\veriler.xml") **Then**

**Return** GetirListe()

**End** **If**

**Dim** serializer **As** XmlSerializer = \_  
 **New** XmlSerializer(**GetType**(KisiListesi))

**Dim** o **As** **Object**

**Dim** f **As** FileStream = **New** FileStream("c:\veriler.xml", FileMode.Open)

o = serializer.Deserialize(f)

f.Close()

**Return** **CType**(o, KisiListesi)

**End** **Function**

**End** **Class**

Kod 409, KisiListesiAdaptoru Sınıf Tanımı

Veri Bağlama

Programcı tarafından tanımlanan sınıfların veri bağlamada kullanımını göstermek amacıyla veri tablosu kontrolü kullanılacaktır.

Ekran Çıktısı 162, Tasarım Anında Veri Tablosu

Butonlara tıklanmasıyla çalıştırılan kodlar aşağıda gösterilmiştir.

C#

**private** **void** ctlBaglaBtn\_Click(**object** sender, EventArgs e) {

ctlGrid.DataSource = KisiListesiAdaptoru.GetirListe();

}

**private** **void** ctlKaydet\_Click(**object** sender, EventArgs e) {

KisiListesiAdaptoru.Kaydet((KisiListesi)ctlGrid.DataSource);

}

**private** **void** ctlYukle\_Click(**object** sender, EventArgs e) {

ctlGrid.DataSource = KisiListesiAdaptoru.Yukle();

}

VB.NET

**Private** **Sub** ctlBaglaBtn\_Click(**ByVal** sender **As** System.**Object**, \_  
 **ByVal** e **As** System.EventArgs) \_   
 **Handles** ctlBaglaBtn.Click

ctlGrid.DataSource = KisiListesiAdaptoru.GetirListe()

**End** **Sub**

**Private** **Sub** ctlKaydet\_Click(**ByVal** sender **As** System.**Object**, \_  
 **ByVal** e **As** System.EventArgs) \_  
 **Handles** ctlKaydet.Click

KisiListesiAdaptoru.Kaydet(**CType**(ctlGrid.DataSource, KisiListesi))

**End** **Sub**

**Private** **Sub** ctlYukle\_Click(**ByVal** sender **As** System.**Object**, \_  
 **ByVal** e **As** System.EventArgs) \_   
 **Handles** ctlYukle.Click

ctlGrid.DataSource = KisiListesiAdaptoru.Yukle()

**End** **Sub**

Kod 410, Programcı Tanımlı Sınıflar ve Veri Bağlama

Veri Bağla komutunun verilmesiyle, KisiListesiAdaptoru nesnesi tarafından üretilen kişi bilgileri veri tablosu kontrolü tarafından gösterilmektedir. Veri tablosunda nesne özelliklerinin otomatik olarak gösterildiğine dikkat ediniz. Kisi sınıfı bu davranışı değiştirmek için ITypedList arabirim gerçekleştirimini yapabilir ve hangi özelliklerin veri bağlamada kullanılacağını belirtebilmektedir.

Ekran Çıktısı 163, Veri Tablosu Kontrolü Veri Bağlama Çıktısı

Veri tablosu kontrolü, gösterilen verinin düzenlenebilmesini sağlamaktadır. Örneğin ilk kayıt değerinin değiştirilmesi ve XML Olarak Kaydet komutunun verilmesiyle, veri değiştirildiği şekliyle XML dosyasına yazılmaktadır.

Ekran Çıktısı 164, Veri Tablosu Kontrolü Üzerinde Verinin Düzenlenmesi

Aşağıda üretilen XML dosyası içeriği gösterilmiştir.

<?xml version="1.0"?>

<ArrayOfKisiBilgisi  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"   
 xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<KisiBilgisi>

<Id>1</Id>

<TcKimlikNo>56443344551</TcKimlikNo>

<Ad>Veli</Ad>

<Soyad>Yılmaz</Soyad>

<DogumTarih>1955-09-09T00:00:00</DogumTarih>

</KisiBilgisi>

<KisiBilgisi>

<Id>2</Id>

<TcKimlikNo>36423140671</TcKimlikNo>

<Ad>Mahmut</Ad>

<Soyad>Önder</Soyad>

<DogumTarih>1945-09-09T00:00:00</DogumTarih>

</KisiBilgisi>

</ArrayOfKisiBilgisi>

XML içeriğinde değiştirilen alan değeri bulunmaktadır. Kullanıcı tarafından veride yapılan değişiklik otomatik olarak veriye yansıtılmıştır.

BindingList Sınıfı

System.ComponentModel.BindingList sınıf tanımı aşağıda gösterilmiştir.

C#

**public** **class** BindingList<T> : Collection<T>, IBindingList, IList,   
 ICollection, IEnumerable, ICancelAddNew,   
 IRaiseItemChangedEvents

VB.NET

**Public** **Class** BindingList(**Of** T)

**Inherits** Collection(**Of** T)

**Implements** IBindingList, IList, ICollection, IEnumerable, \_  
 ICancelAddNew, IRaiseItemChangedEvents

Kod 411, System.ComponentModel.BindingList Sınıf Tanımı

Programcı tanımlı sınıfların veri bağlama desteğinin sağlanması amacıyla jenerik bir sınıf olan BindingList sınıfı kullanılabilir. Bu sınıf temel veri bağlama işlevlerini desteklemektedir.

Örneğin, bir önceki konu başlığında gösterilen KisiListesi nesnesinin veri tablosu kontrolüne bağlanması sonucu, kontrol tarafından mevcut veri üzerinde düzenlemeler yapılabilmesine rağmen yeni kayıtlar eklenememektedir. List sınıfını miras alan KisiListesi sınıfının BindingList sınıfını miras alan sürümü aşağıda gösterilmiştir.

C#

[Serializable]

**public** **class** KisiTablosu : BindingList<KisiBilgisi> {

}

VB.NET

<Serializable()> \_

**Public** **Class** KisiTablosu

**Inherits** BindingList(**Of** KisiBilgisi)

**End** **Class**

Kod 412, BindingList Kullanım Örneği

BindingList sınıfının sağlamış olduğu temel işlevler, bu sınıfı miras alan sınıflarla yapılan veri bağlamanın kolaylıkla yönetimini sağlamaktadır.

Verinin seçilmesi ve kaydedilmesi amacıyla kullanılabilecek KisiTablosuAdaptoru isimli sınıf aşağıda gösterilmiştir.

C#

**public** **class** KisiTablosuAdaptoru {

**private** **static** KisiTablosu liste = **null**;

**public** **static** KisiTablosu GetirListe() {

**if** (liste == **null**)

liste = **new** KisiTablosu();

liste.Add(**new** KisiBilgisi(1, 56443344551, "Mehmet", "Yılmaz", \_  
 **new** DateTime(1955, 9, 9)));

liste.Add(**new** KisiBilgisi(2, 36423140671, "Mahmut", "Önder", \_  
 **new** DateTime(1945, 9, 9)));

**return** liste;

}

**public** **static** **void** Kaydet(KisiTablosu liste) {

XmlSerializer serializer = **new** XmlSerializer(**typeof**(KisiTablosu));

**using** (FileStream f = **new** FileStream(@"c:\veriler.xml", FileMode.Create)) {

serializer.Serialize(f, liste);

f.Close();

}

}

**public** **static** KisiTablosu Yukle() {

**if** (!File.Exists(@"c:\veriler.xml"))

**return** GetirListe();

XmlSerializer serializer = **new** XmlSerializer(**typeof**(KisiTablosu));

**object** o;

**using** (FileStream f = **new** FileStream(@"c:\veriler.xml", FileMode.Open)) {

o = serializer.Deserialize(f);

f.Close();

}

**return** (KisiTablosu)o;

}

}

VB.NET

**Public** **Class** KisiTablosuAdaptoru

**Private** **Shared** liste **As** KisiTablosu = **Nothing**

**Public** **Shared** **Function** GetirListe() **As** KisiTablosu

**If** liste **Is** **Nothing** **Then**

liste = **New** KisiTablosu

**End** **If**

liste.Add(**New** KisiBilgisi(1, 56443344551, "Mehmet", "Yılmaz", \_  
 **New** DateTime(1955, 9, 9)))

liste.Add(**New** KisiBilgisi(2, 36423140671, "Mahmut", "Önder", \_  
 **New** DateTime(1945, 9, 9)))

**Return** liste

**End** **Function**

**Public** **Shared** **Sub** Kaydet(**ByVal** liste **As** KisiTablosu)

**Dim** serializer **As** XmlSerializer = \_  
 **New** XmlSerializer(**GetType**(KisiTablosu))

**Dim** f **As** FileStream = **New** FileStream("c:\veriler.xml", FileMode.Create)

serializer.Serialize(f, liste)

f.Close()

**End** **Sub**

**Public** **Shared** **Function** Yukle() **As** KisiTablosu

**If** **Not** File.Exists("c:\veriler.xml") **Then**

**Return** GetirListe()

**End** **If**

**Dim** serializer **As** XmlSerializer = \_  
 **New** XmlSerializer(**GetType**(KisiTablosu))

**Dim** o **As** **Object**

**Dim** f **As** FileStream = **New** FileStream("c:\veriler.xml", FileMode.Open)

o = serializer.Deserialize(f)

f.Close()

**Return** **CType**(o, KisiTablosu)

**End** **Function**

**End** **Class**

Kod 413, KisiTablosuAdaptoru Sınıfı

Kullanıcı ara yüz kontrolüne KisiListesi yerine KisiTablosu nesnesinin bağlanmasıyla yeni kayıtlar da kolaylıkla eklenebilmektedir.

Aşağıdaki ekran çıktısında yeni kayıtlar eklenmiş veri tablosu kontrolü gösterilmektedir.

Ekran Çıktısı 165, BindingList Sınıfı ve Veri Bağlama

Kullanıcının veri üzerinde yaptığı değişiklik ve eklemeler veri tablosu kontrolü tarafından, .NET veri bağlama mimarisi kullanılarak, KisiTablosu nesnesine iletilmektedir. Aşağıda bir kayıt eklenerek verinin kaydedilmesi sonucu üretilen XML dosyası içeriği gösterilmiştir.

<?xml version="1.0"?>

<ArrayOfKisiBilgisi  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"   
 xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<KisiBilgisi>

<Id>1</Id>

<TcKimlikNo>56443344551</TcKimlikNo>

<Ad>Mehmet</Ad>

<Soyad>Yılmaz</Soyad>

<DogumTarih>1955-09-09T00:00:00</DogumTarih>

</KisiBilgisi>

<KisiBilgisi>

<Id>2</Id>

<TcKimlikNo>36423140671</TcKimlikNo>

<Ad>Mahmut</Ad>

<Soyad>Önder</Soyad>

<DogumTarih>1945-09-09T00:00:00</DogumTarih>

</KisiBilgisi>

<KisiBilgisi>

<Id>3</Id>

<TcKimlikNo>45332233432</TcKimlikNo>

<Ad>Hasan</Ad>

<Soyad>Çalışkan</Soyad>

<DogumTarih>1977-01-01T00:00:00</DogumTarih>

</KisiBilgisi>

</ArrayOfKisiBilgisi>

Veri küme ve tablolarıyla yapılan veri bağlama ile yukarıda genel özellikleriyle aktarılan programcı tanımlı sınıfların veri bağlamasının aynı kurallarla yönetildiğine dikkat ediniz. Gösterilen yöntemler, uygulama yazılımlarında iş varlıkları olarak veri küme ve tablolarının tercih edilmediği durumlarda rahatlıkla verinin ara yüz kontrollerine bağlanabilmesi amacıyla kullanılabilmektedir.

İnsanoğullarının yönetimi sanatı üzerinde düşünen herkes, devletlerin geleceğinin gençlerinin eğitimine bağlı olduğu konusunda ikna olmuşlardır.

Aristoles

Bölüm

28

VS.NET ve Veri Kullanım Yöntemleri

V

eri bağlama mimarisi ve kuralları hakkında verilen teorik bilgilerin ardından bu bölümde VS.NET ile veri bağlama temelleri ve çeşitli veri bağlama yöntemleri hakkında bilgiler verilecektir.

Genel Bilgiler

Ara yüz katmanı tarafından verinin alınması, görüntülenmesi ve kullanıcı tarafından yapılan düzenlemelerin veri kaynağına iletimi farklı yöntemlerle yapılabilmektedir. Tercih edilen yöntemler iki temel kategori çerçevesinde ele alınabilir.

* Ara yüz katmanından direkt veritabanı erişimi sağlanarak verinin elde edilmesi.
* Ara yüz katmanından iş nesneleri erişimi sağlanarak verinin elde edilmesi.

Direkt Veritabanı Erişimi

Bilindiği gibi çok katmanlı mimari kuralları çerçevesinde kullanıcı ara yüz katmanından veri katmanına direkt erişim önerilmemektedir. Buna rağmen, iş katman gerçekleştiriminin yapılmasının tercih edilmediği küçük projelerde bu yöntem uygulanabilmekte ve yazılım geliştirme süresini azaltabilmektedir.

İş Nesneleri Kullanımı

İş katmanı içerisinde bulunan nesnelerin ara yüz katmanına veriyi iletmeleri ve tüm iş mantığının bu katman içerisinde gerçekleştirimi önerilen yöntemdir.

ASP.NET ve Windows Uygulamaları

.NET veri bağlama mimarisi aynı olmasına rağmen, ASP.NET ve Windows uygulamalarında verinin elde edilmesi amacıyla kullanılan yöntemler farklılık göstermektedir. Bunun en önemli sebebi, web uygulama mimarisi ile Windows uygulama mimarisi arasındaki temel yöntem farklılıklarıdır.

Web uygulamaları http üzerinden istek – sonuç (ing. Request – Response) temelli çalışıyor olmasına rağmen Windows uygulamaları çalıştıklarında bilgisayar hafızasına yüklenmektedir.

MVC Tasarım Şablonu

Model-View-Controller tasarım şablonu üç temel yapıdan oluşmaktadır.

* Model, veriyi ifade etmektedir.
* View, verinin görüntülendiği ara yüz kontrollerini ifade etmektedir.
* Controller, model ve veri arasındaki ilişki yönetimini ifade etmektedir.

VS.NET ile yapılan görsel veri bağlamanın temelinde MVC tasarım şablonu kullanılmaktadır. Şablonun gerçekleştiriminde kullanılan sınıf ve yöntemler hakkında Windows Forms.NET, ASP.NET ve PDA uygulamaları bazında bilgiler sunulacaktır.

Windows Forms.NET Uygulamaları

Windows Forms.NET uygulamalarında görsel veri bağlamanın yapılması amacıyla VS.NET tarafından kullanımı kolay ve pratik yöntemler sunulmaktadır.

Bu konu başlığında, daha önce aktarılan bilgiler temel alınarak veri bağlamanın görsel olarak yapılması hakkında bilgiler sunulacaktır.

Veri Kaynakları

Veri kaynakları (ing. Data Sources), Windows Forms.NET projelerinde veriye erişmek ve görsel olarak veri bağlamanın yapılabilmesini sağlamaktadırlar.

Veri Kaynakları Penceresi

Data | Show Data Sources komutu ile görüntülenebilen VS.NET veri kaynakları penceresi ekran çıktısı aşağıda gösterilmiştir.

Ekran Çıktısı 166, VS.NET Veri Kaynakları Penceresi

Veri kaynakları penceresi, istenilen verinin tasarım ortamında görüntülenebilmesi ve sürükle – bırak (ing. Drag – Drop) yöntemiyle kullanıcı ara yüz kontrollerinin otomatik olarak oluşturulabilmesini sağlamaktadır.

Yeni bir veri kaynağı eklemek için veri Data | Add New Data Source komutu verilmelidir.

Komutun verilmesiyle, eklenmek istenen veri kaynağının veriyi nereden elde etmesi gerektiğinin seçimi yapılmaktadır.

* Database, veritabanından veri erişimi için kullanılması gereken seçenektir.
* Web Service, web hizmetinden verinin elde edilmesi için kullanılması gereken seçenektir.
* Object, verinin istenilen nesneden elde edilmesi için kullanılması gereken seçenektir.

Aşağıdaki ekran çıktısında seçimin yapıldığı pencere gösterilmiştir.

Ekran Çıktısı 167, Veri Kaynak Tipi Seçimi

Veritabanı Kaynağı

Veritabanı kaynağı seçeneği ile yapılan işlem veritabanı sihirbazının çalıştırılmasıdır. İlk adımda hangi veritabanının kullanılmak istendiği ve veritabanı bağlantı dizesi seçimi yapılmaktadır.

Ekran Çıktısı 168, Veritabanı Bağlantısının Seçimi

Veritabanı içerisinde bulunan tablo ve depolanmış prosedürlerin seçimi bir sonraki pencere tarafından yapılmaktadır. Sihirbaz, kitabın daha önceki bölümlerinde aktarılan tablo adaptör sihirbazına benzerlik göstermektedir.

Aşağıdaki ekran çıktısında veriye erişim için kullanılması istenen veritabanı yapıları gösterilmiştir.

Ekran Çıktısı 169, Veritabanı Yapılarının Seçimi

Sihirbazın tamamlanmasıyla yapılan işlem, projeye yeni bir veri kümesi eklenmesi ve seçilen tablolar için tablo, adaptör, ilişki ve sorgu sınıflarının oluşturulmasıdır.

Ekran Çıktısı 170, Sihirbaz Tarafından Oluşturulan Veri Kümesi

Aynı işlemin programcı tarafından projeye veri kümesi eklenmesi ve VS.NET veri kümesi editörü ile istenilen yapının oluşturulmasıyla sağlanması da mümkündür.

İşlemin tamamlanması sonucunda veri kaynakları penceresi ekran çıktısı aşağıda gösterilmiştir.

Ekran Çıktısı 171, Veri Kaynakları Penceresi ve Veriler

Web Hizmet Kaynağı

Web hizmet kaynağının seçilmesiyle yapılan işlem, önceki bölümde aktarılan web hizmet referansının projeye eklenmesidir.

Ekran Çıktısı 172, Web Hizmet Referansı Eklenmesi

Add Reference komutunun verilmesiyle eklenen web hizmet referansı aşağıda gösterilmiştir.

Ekran Çıktısı 173, Web Hizmet Referansı

Veri kaynakları penceresi projede bulunan veri kaynaklarını otomatik olarak listelemektedir. Aşağıdaki ekran çıktısında eklenen web hizmet kaynağı gösterilmiştir.

Ekran Çıktısı 174, Web Hizmet Kaynağı ve Veri Kaynakları Penceresi

Nesne Kaynağı

Nesne kaynağı, istenilen Assembly dosyası içerisinde bulunan nesnelerin veri kaynağı olarak projeye eklenebilmesini sağlamaktadır.

Ekran Çıktısı 175, Veri Kaynak Nesnesi Seçim Penceresi

İşlemin tamamlanmasıyla elde edilen veri kaynakları ve VS.NET çözüm penceresi ekran çıktıları aşağıda gösterilmiştir.

Ekran Çıktısı 176, Veri Kaynakları ve VS.NET Çözüm Penceresi

Veri Kontrolleri Kullanımı

Ekran çıktılarından da görüleceği üzere, veri kaynakları penceresi açısından değerlendirildiğinde verinin hangi kaynaktan geldiğinin herhangi bir önemi bulunmamaktadır. Veri hangi kaynaktan gelirse gelsin, .NET veri bağlama mimarisi kuralları uyarınca nesneler sorgulanmakta ve nesneler tarafından iletilen özellikler pencerede görüntülenmektedir.

System.Windows.Forms.BindingSource, model ile ara yüz kontrolleri arasında veri bağlamayı sağlayan sınıftır. Bir başka anlatımla, BindingSource, verinin hangi kaynaktan geldiğinden bağımsız olarak ara yüz kontrolleri ve veri kaynağı arasındaki köprüyü sağlamaktadır.

Şekil 60, Veri Bağlama Kaynağı ve Veri Bağlama

BindingSource sınıf tanımı aşağıda gösterilmiştir.

| C# | VB.NET |
| --- | --- |
| **public** **class** BindingSource :  Component, IBindingListView, IBindingList, IList, ICollection, IEnumerable, ITypedList, ICancelAddNew, ISupportInitializeNotification, ISupportInitialize, ICurrencyManagerProvider | **Public** **Class** BindingSource  **Inherits** Component  **Implements** IBindingListView, \_ IBindingList, IList, ICollection, \_ IEnumerable, ITypedList, \_ ICancelAddNew, \_ ISupportInitializeNotification, \_ ISupportInitialize, \_ ICurrencyManagerProvider |

Kod 414, System.Windows.Forms.BindingSource Sınıf Tanımı

Tanımdan, BindingSource sınıfının .NET veri bağlama mimarisi tarafından belirlenen arabirim gerçekleştirimlerini yaptığı görülmektedir. Sınıfın System.ComponentMode.Component sınıfını miras alması tasarım anında kullanılabilmesini sağlamaktadır.

Sınıfın ICurrencyManagerProvider arabirim gerçekleştirimini yapması ile Windows Forms.NET uygulamalarında aktif kayıt kavramı bağlantısız modelde kullanılabilmektedir.

Veri Kontrolleri

Windows Forms.NET uygulamalarında kullanılabilecek temel ara yüz kontrolleri ve açıklamaları aşağıda gösterilmiştir.

| Kontrol | Açıklama |
| --- | --- |
| DataGridView | Verinin tablo şeklinde görüntülendiği ve kullanıcı tarafından düzenlemelerin yapılabileceği kontroldür. |
| BindingNavigator | Veri ekleme ve veriler arasında dolaşmak için kullanılan kontroldür. |
| ReportViewer | Veri raporlama amacıyla kullanılan kontroldür. |

Tablo 157, Windows Forms.NET Ara Yüz Kontrolleri ve Açıklamaları

Sürükle – Bırak Yöntemi

Veri kaynakları penceresinde bulunan verinin form üzerine sürüklenmesiyle VS.NET tarafından otomatik olarak gerekli kontroller form üzerinde oluşturulmakta ve özellikleri belirlenmektedir.

Aşağıdaki ekran çıktısında, veri kaynakları penceresinden sürükle – bırak yöntemi ile oluşturulan kullanıcı ara yüz kontrolleri gösterilmiştir.

Ekran Çıktısı 177, Sürükle - Bırak Yöntemi ile Kontrollerin Oluşturulması

Oluşturulan kontroller ve açıklamaları aşağıda gösterilmiştir.

| Kontrol | Açıklama |
| --- | --- |
| kisiGridView | Kişi bilgilerinin görüntülendiği ve düzenlenebileceği veri tablo kontrolüdür. |
| kitapDataSet | Veri kümesi nesnesidir. |
| kisiBindingSource | Veri kaynağı ve ara yüz kontrolleri arasındaki köprü olan veri bağlama kaynağı kontrolüdür. |
| kisiTableAdaptor | Kişi tablo adaptör nesnesidir. |
| kisiBindingNavigator | Kayıtlar arasında dolaşmak için kullanılan kontroldür. |

Tablo 158, Oluşturulan Kontrolleri ve Açıklamaları

Verinin yüklenmesi için gerekli kod parçası aşağıda gösterilmiştir.

C#

**this**.kisiTableAdapter.Fill(**this**.kitapDataSet.Kisi);

VB.NET

**Me**.kisiTableAdapter.Fill(**Me**.kitapDataSet.Kisi)

Kod 415, Verinin Yüklenmesi

Kod parçasında dikkat edilmesi gereken konulardan birisi de kullanıcı ara yüz kontrollerine herhangi bir veri bağlamanın programcı tarafından yapılmadığıdır. Programcı tarafından yapılan işlem, kitabın daha önceki konularına aktarılan şekilde veri tablosu nesnesinin adaptör ile doldurulmasıdır. Veri tablosunun doldurulmasıyla birlikte otomatik olarak kullanıcı ara yüz kontrollerine veri bağlanmaktadır. Veri bağlama, kisiBindingSource tarafından verinin değişmesiyle birlikte otomatik olarak yapılmaktadır.

Verinin yüklenmesi sonucu elde edilen ekran çıktısı aşağıda gösterilmiştir.

Ekran Çıktısı 178, Otomatik Veri Bağlama ve Veriler

Sürükle – Bırak yönetimi ile oluşturulan ara yüz kontrollerinin bağlanan veri ve veri özellikleri bazında özelleştirilebilmesi mümkündür.

Ekran Çıktısı 179, Ara Yüz Kontrollerinin Özelleştirilmesi

Ara Yüz Kontrolleri

Veri bağlama amacıyla kullanılan ara yüz kontrolleri temel iki kategoride incelenebilir.

* Veri tablo kontrolü, liste kutusu gibi veri kaynağı verisinin tamamını içeren kontroller.
* Aktif kayıt içerisinde bulunan alan değerini içeren kontroller.

Aşağıdaki ekran çıktısında veri tablo kontrolü DataSource özelliği tasarım anında gösterilmiştir.

Ekran Çıktısı 180, Veri Tablosu Kontrolü ve Veri Kaynağı

Kontrol özelliklerine istenilen verinin bağlanabilmesini bütün ara yüz kontrollerinin miras aldığı Control.DataBindings özelliği sağlamaktadır.

C#

**public** ControlBindingsCollection DataBindings { **get**; }

VB.NET

**Public** **ReadOnly** **Property** DataBindings **As** ControlBindingsCollection

Kod 416, System.Windows.Forms.Control.DataBindings Tanımı

Tanımdan da görüleceği üzere koleksiyon sınıf tipinde olan bu özellik yardımıyla, kontrolün farklı özelliklerine farklı verilerin bağlanabilmesi mümkün olmaktadır.

Buton kontrolü olan Button sınıfının özelliklerine çalışma zamanında veri bağlama aşağıdaki kod parçasında gösterilmiştir.

C#

ctlTestBtn.DataBindings.Add("Text", kisiBindingSource, "Ad");

VB.NET

ctlTestBtn.DataBindings.Add("Text", kisiBindingSource, "Ad")

Kod 417, Control.DataBindings Özelliği Kullanımı

Kod parçası ile yapılan işlem, buton kontrolü Text özelliğinin, kisiBindingSource nesnesi Ad alanına bağlanmasıdır.

Ekran Çıktısı 181, Buton Kontrolüne Veri Bağlanması

Kontrol özelliklerine bağlanmak istenen verinin tasarım anında belirtilmesi de mümkündür.

Ekran Çıktısı 182, Tasarım Anında Kontrol Özellikleri Veri Bağlama

Veri Bağlama Kaynağı Kullanımı

Veri bağlama kaynağı kontrolünün sürükle – bırak yöntemi ile otomatik olarak atanan özellik değerleri aşağıda gösterilmiştir.

Ekran Çıktısı 183, Veri Bağlama Kaynağı Özellikleri

BindingSource.DataSource özelliği, veri bağlama kaynağının veriyi alacağı nesneyi belirtmektedir. Ekran çıktısında dikkat edilmesi gereken önemli bir konu, DataSource özelliğine atanan nesnenin form üzerinde bulunan veri kümesi nesnesi olduğudur. Project Data Sources – KitapDataSet sadece tip tanımını ifade etmektedir. Tip tanımı ile, tip tarafından gerçekleştirimi yapılan veri bağlama arabirimleri yardımıyla, tip özellikleri elde edilmektedir.

kitapDataSet ismindeki nesnenin form üzerinde bulunmasıyla, verinin bağlanması için yeni bir nesne oluşturulması gerekmemektedir.

Örneğin programcı tanımlı KisiTablosu tipinin sürükle – bırak yöntemi kullanılarak forma eklenerek elde edilmiş ekran çıktısı aşağıda gösterilmiştir.

Ekran Çıktısı 184, Proje Veri Kaynakları ve Veri Bağlama

Burada yapılan işlem, nesnenin kendisi yerine Model.Tipler.KisiTablosu tipinin veri kaynağı olarak belirtilmesidir. Böylece, tasarım anında sınıf özelliklerine erişim mümkün olmaktadır.

Çalışma zamanında verinin bağlanması aşağıdaki kod parçasında gösterilmiştir.

C#

KisiTablosu tablo = KisiTablosuAdaptoru.GetirListe();

kisiTablosuBindingSource.DataSource = tablo;

VB.NET

KisiTablosu tablo = KisiTablosuAdaptoru.GetirListe()

kisiTablosuBindingSource.DataSource = tablo

Kod 418, Çalışma Zamanında Nesne Bağlama

KisiTablosu.GetirListe metodunun KisiTablosu nesnesi döndürdüğüne dikkat edilmelidir.

Ekran Çıktısı 185, Programcı Tanımlı Veri Bağlama Sonucu

Üye Bilgileri

Aşağıda BindingSource sınıfı üye ve açıklamaları gösterilmiştir.

| Üye Adı | Açıklama |
| --- | --- |
| AllowEdit | Veri kaynağı verilerinin değiştirilip değiştirilemeyeceğini belirtir. |
| AllowNew | Veri kaynağına yeni veri eklenip eklenemeyeceğini belirtir. |
| Count | Veri kaynağı eleman (kayıt) sayısını belirtir. |
| CurrencyManager | Aktif kayıt yönetimini yapan nesneyi döndürür. |
| Current | Aktif kaydı döndürür. |
| Filter | Veri kaynağına iletilen filtreleme değerini belirtir. |
| List | Veri kaynağını belirtir. |
| Position | Aktif kayıt pozisyon bilgisini belirtir. |
| CancelEdit | Aktif kayıt üzerinde yapılan değişikliği iptal eder. |
| EndInit | Kullanıcı tarafından aktif kayıt üzerinde yapılan değişiklikleri kabul eder. |
| MoveFirst | İlk kayda konumlanır. |
| MoveLast | Son kayda konumlanır. |
| MoveNext | Sonraki kayda konumlanır. |
| MovePrevious | Önceki kayda konumlanır. |

Tablo 159, System.Windows.Forms.BindingSource Sınıfı Üye Bilgileri

Olaylar

Veri bağlama kaynağı olayları kullanılarak, kullanıcı tarafından veri üzerinde yapılan değişiklikler, veri bağlama süreci ve oluşan hataların yönetimi yapılabilmektedir.

Veri kaynağı olayları ve açıklamaları aşağıda gösterilmiştir.

| Olay Adı | Açıklama |
| --- | --- |
| AddingNew | Yeni bir kayıt eklenmeden hemen önce tetiklenen olaydır. |
| BindingComplete | Veri bağlama işlemleri tamamlandığında tetiklenen olaydır. |
| CurrentChanged | Aktif kayıt değiştiğinde tetiklenen olaydır. |
| CurrentItemChanged | Aktif kaydın herhangi bir özelliği değiştiğinde tetiklenen olaydır. |
| DataError | Veri bağlamada oluşan hata durumunda tetiklenen olaydır. |
| DataMemberChanged | BindingSource.DataMember özelliği değiştiğinde tetiklenen olaydır. |
| DataSourceChanged | Veri kaynağı değiştiğinde tetiklenen olaydır. |
| ListChanged | Veri bağlama kaynağının veri kaynağı olarak kullandığı liste değiştiğinde tetiklenen olaydır. |
| PositionChanged | Aktif pozisyon değiştiğinde tetiklenen olaydır. |

Tablo 160, Veri Bağlama Kaynağı Olayları ve Açıklamaları

Rapor Kontrolü Kullanımı

Microsoft.Reporting.WinForms.ReportViewer kontrolü Windows Forms.NET uygulamalarında rapor tanımlama dili (Report Definition Language – RDL) kullanılarak hazırlanmış olan raporların görüntülenmesi ve farklı formatlarda dışa aktarımı için kullanılmaktadır.

RDL tarafından sağlanan işlev rapor görselliğinin tanımlanabilmesidir. .NET veri bağlama mimarisi ile, rapor görselliği ve veri birleştirilmekte, böylece rapor oluşturulmaktadır. Rapor kontrolü gerek yerel gerek Microsoft Reporting Services yazılımı ile üretilen uzak raporları gösterebilmektedir.

Raporlama geniş ve detaylı bir konudur. Bu konu başlığında, temel düzeyde ve okuyucunun raporlama konusunda bilgisini artırabilmesi için gerekli yolu açmak amacıyla örnek rapor hazırlanması gösterilecektir.

Windows rapor kontrolü Microsoft.ReportViewer.WinForms.dll isimli Assembly dosyası içerisinde tanımlanmıştır.

Rapor Eklenmesi ve Tasarlanması

Yeni bir rapor eklemek için proje üzerinde sağ tıklayarak Add | New Item … komutu veriniz.

Ekran Çıktısı 186, Projeye Rapor Eklenmesi

Veri kaynakları penceresi yardımıyla istenilen verinin rapor üzerine sürüklenerek görsel tasarımın yapılması mümkündür.

Ekran Çıktısı 187, Rapor Tasarımı

Rapor Görüntüleyicisi

Rapor görüntüleyici kontrolü ve tasarlanan raporun görüntüleyiciye iletimi aşağıda gösterilmiştir.

Ekran Çıktısı 188, Rapor Görüntüleyicisi ve Rapor Seçimi

Rapor Veri Kaynağının Belirtilmesi

Raporun oluşturulması için rapor verisinin belirtilmesi gerekmektedir. Bu işlemin tasarım anında yapılabilmesi için Choose Data Sources komutu verilerek veri bağlamada kullanılmak istenen nesne belirtilebilir.

Ekran Çıktısı 189, Rapor Veri Kaynağının Belirtilmesi

Tasarım anında yapılan işlemin çalışma zamanında yapılabilmesi için gerekli kod parçası aşağıda gösterilmiştir.

C#

ctlRaporGotuntuleyici.LocalReport.DataSources.Clear();

KitapDataSet ds = **new** KitapDataSet();

kisiTableAdapter.Fill(ds.Kisi);

ReportDataSource kaynak = **new** ReportDataSource("KitapDataSet\_Kisi", ds.Kisi);

ctlRaporGotuntuleyici.LocalReport.DataSources.Add(kaynak);

VB.NET

ctlRaporGotuntuleyici.LocalReport.DataSources.Clear()

**Dim** ds **As** KitapDataSet = **New** KitapDataSet()

kisiTableAdapter.Fill(ds.Kisi)

**Dim** kaynak **As** ReportDataSource = \_  
 **New** ReportDataSource("KitapDataSet\_Kisi", ds.Kisi)

ctlRaporGotuntuleyici.LocalReport.DataSources.Add(kaynak)

Kod 419, Çalışma Zamanında Rapor Veri Kaynağının Belirtilmesi

Raporun Görüntülenmesi

Veri kaynakları gösterildiği şekilde düzenlendikten sonra raporun görüntülenmesi için ReportViewer.RefreshReport metodunun çağrılması yeterlidir.

Rapor çıktısı aşağıda gösterilmiştir.

Ekran Çıktısı 190, Örnek Rapor Çıktısı

ASP.NET Web Uygulamaları

ASP.NET web uygulamalarında verinin ara yüz kontrollerine bağlanması ve düzenlenmesi için, Windows Forms.NET uygulamalarına benzer şekilde, veri kaynağı kontrolleri kullanılmaktadır.

Veri Kaynağı Kontrolleri

Veri kaynağı kontrolleri (ing. DataSource Control), ara yüz kontrolleri ile veri arasında bir katman oluşturmaktadır. Oluşturulan bu katman yardımıyla farklı ara yüz kontrollerinin aynı veri kaynağı kontrollerine bağlanabilmesi ve böylece verinin tek noktadan yönetilebilmesi mümkün olmaktadır.

Veri kaynağı kontrolleri System.Web.UI.DataSourceControl soyut sınıfını miras almaktadır.

C#

**public** **abstract** **class** DataSourceControl : Control,   
 IDataSource, IListSource

VB.NET

**Public** **MustInherit** **Class** DataSourceControl

**Inherits** Control

**Implements** IDataSource, IListSource

Kod 420, System.Web.UI.DataSourceControl Sınıf Tanımı

Sınıf tarafından miras alınan System.Web.UI.Control sınıfı, tüm ASP.NET sunucu kontrolleri (ing. Server Control) için temel ata sınıftır. Control sınıfının miras alınması veri kaynağı kontrolünün tasarım zamanında kullanılabilmesini sağlamaktadır.

Kitabın daha önceki bölümlerinde verilen bilgilerden hatırlanacağı üzere System.ComponentModel.IListSource sınıfı, .NET veri bağlama mimarisi kapsamında kullanılmaktadır. Sınıfın IListSource arabirim gerçekleştirimini yapması durumunda, veri bağlama işleminde nesnenin kendisi yerine nesne tarafından döndürülen liste veya listeler veri bağlamada kullanılmaktadır.

IDataSource arabirim tanımı aşağıda gösterilmiştir.

C#

**public** **interface** IDataSource {

**event** EventHandler DataSourceChanged;

DataSourceView GetView(**string** viewName);

ICollection GetViewNames();

}

VB.NET

**Public** **Interface** IDataSource

**Event** DataSourceChanged **As** EventHandler

**Function** GetView(**ByVal** viewName **As** **String**) **As** DataSourceView

**Function** GetViewNames() **As** ICollection

**End** **Interface**

Kod 421, System.Web.UI.IDataSource Arabirim Tanımı

IDataSource.DataSourceChanged metodu, verinin değişmesi sonucu tetiklenen olayı ifade etmektedir. IDataSource.GetView metodu, parametre olarak iletilen değer adında olan veri kaynağı görünümüne (ing. DataSource View) erişmek için kullanılmaktadır. IDataSource.GetViewNames metodu, veri kaynağı kontrollü içerisinde bulunan görünümleri döndürmektedir.

Arabirim tanımından, bir veri kaynağı kontrolünün içerisinde farklı görünümler bulunabileceği anlaşılmaktadır. Bu durum veri kümeleri ve tabloları ile benzerlik göstermektedir. Nasıl ki bir veri kümesi içerisinde birden çok tablo bulunabiliyorsa, veri kaynağı kontrolü içersinde de birden çok görünüm bulunabilmektedir.

Genel mimari aşağıda şematik olarak gösterilmiştir.

Şekil 61, Veri Kaynağı Kontrol ve Görünümü

Veri kaynağı kontrolü, ara yüz kontrolü ile veri kaynağı görünümü arasında köprü görevi yapmaktadır.

Veri Kaynağı Görünümleri

Veri kaynağı görünümleri soyut bir sınıf olan System.Web.UI.DataSourceView sınıfını miras alan sınıflardır.

C#

**public** **abstract** **class** DataSourceView {

}

VB.NET

**Public** **MustInherit** **Class** DataSourceView

**End Class**

Kod 422, System.Web.UI.DataSourceView Sınıf Tanımı

Tasarımdan da görülebileceği gibi veri görünüm sınıfı herhangi bir sınıfı miras almamakta veya arabirim gerçekleştirimi yapmamaktadır.

Sınıf üye bilgileri aşağıda gösterilmiştir.

| Üye Adı | Açıklama |
| --- | --- |
| CanDelete | Görünümün veri silme işlemini destekleyip desteklemediğini belirtir. |
| CanInsert | Görünümün veri ekleme işlemini destekleyip desteklemediğini belirtir. |
| CanPage | Görünümün veri sayfalamayı destekleyip desteklemediğini belirtir. |
| CanRetrieveTotalRowCount | Görünümün toplam kayıt sayısını alıp alamayacağını belirtir. |
| CanSort | Görünümün sıralama yapıp yapamayacağını belirtir. |
| CanUpdate | Görünümün veri güncelleme işlemini destekleyip desteklemediğini belirtir. |
| Name | Görünüm adını belirtir. |
| Delete | Eleman (kayıt) silmek için kullanılmalıdır. |
| Insert | Yeni kayıt eklemek için kullanılmalıdır. |
| Select | Görünümden veri seçmek için kullanılmalıdır |
| Update | Kayıt güncellemek için kullanılmalıdır. |

Tablo 161, System.Web.UI.DataSourceView Sınıfı Üye Bilgileri

Programcıların DataSourceControl ve DataSourceView sınıflarını miras alarak kendi veri kaynağı kontrol ve görünümlerini oluşturmaları ve veri bağlamada kullanmaları mümkündür.

ASP.NET web uygulamalarında kullanılabilecek veri kaynağı kontrolleri araç kutusu Data sekmesinde bulunmaktadır.

Ekran Çıktısı 191, ASP.NET Veri Kontrolleri

SQL Veri Kaynağı Kontrolü

System.Web.UI.WebControls.SqlDataSource sınıfı, ADO.NET veri sağlayıcıları kullanarak veriye erişim sağlayan veri kaynağı kontrolüdür. Kontrol tarafından yapılan işlem, ProviderName özelliği ile ifade edilen veri sağlayıcısına SQL cümleleri iletmek ve ara yüz kontrollerine verinin bağlanmasını sağlamaktır.

SQL veri kaynağı kontrolü, kullanıcı ara yüz katmanından veri katmanına direkt erişim sağlamaktadır. İş katman gerçekleştiriminin tercih edilmediği küçük çaplı yazılım projelerinde kontrolün kullanımı öneriliyor olsa da, büyük çaplı yazılım projelerinde ara yüz katmanı ile veri katmanı arasında iş katmanının bulunması tercih edilmelidir.

ASP.NET web sayfasında, VS.NET sunucu gezgini penceresi yardımıyla sürükle – bırak yöntemi kullanılarak SQL veri kaynağının oluşturulması ve özelliklerinin otomatik olarak belirlenmesi mümkündür.

Ekran Çıktısı 192, Sürükle - Bırak Yöntemi ve SqlDataSource

Sürükle – bırak yöntemi ile web sayfasına veri tablo (GridView) ve SqlDataSource kontrolleri eklenmekte, kontrol özellikleri otomatik olarak düzenlenmektedir.

Veri tablo kontrolüne kayıt düzenleme ve silme özelliklerinin eklenmesi aşağıdaki ekran çıktısında gösterilmiştir.

Ekran Çıktısı 193, Veri Tablosu Kayıt Düzenleme ve Silme

Sayfanın kaydedilmesi ve internet tarayıcısında görüntülenmesi ile, herhangi bir ek işleme gerek kalmadan, kişi kayıtları veri tablosu kontrolü içerisinde listelenmektedir.

Sürükle – bırak yöntemi ile yapılan işlem seçilen tablo bazında SQL cümlelerinin otomatik olarak üretimini içermektedir. Sihirbaz yardımıyla CRUD işlemleri için kullanılması gereken SQL cümleleri ve parametrelerinin belirtilmesi mümkündür.

Aşağıdaki ekran çıktısında web sayfası içerisinde kişi tablosu ad ve soyadı alanında arama yapmak için kullanılacak metin girdi kontrolleri ve verinin listelenmesini sağlayan veri tablosu kontrolü gösterilmiştir.

Ekran Çıktısı 194, SqlDataSource Yapılandırması ve Ara Yüz Kontrolleri

Configure Data Source… komutu verilerek çalıştırılan sihirbaz, veritabanı bağlantısının seçimi ile başlamaktadır.

Ekran Çıktısı 195, Veritabanı Bağlantısının Seçilmesi

Verinin elde edilmesi ve ekleme, silme ve güncelleme işlemlerinde direkt tablolar tercih edilebileceği gibi, SQL cümlelerinin veya depolanmış prosedürlerin sihirbaz ile seçilmesi de mümkündür.

Bir sonraki sihirbaz sayfasında Specify a custom SQL statement or Stored Procedure seçeneği işaretlenerek seçim yapılabilmektedir.

Ekran Çıktısı 196, Veri Erişim Yönteminin Belirlenmesi

CRUD işlemleri için kullanılmak istenen SQL cümlesinin veya depolanmış prosedürün seçimi yapılabilmektedir.

Ekran Çıktısı 197, CRUD İşlemleri için Depolanmış Prosedürlerin Seçimi

GuncelleKisi, EkleKisi ve SilKisi isimli depolanmış prosedürler sırasıyla Update, Insert ve Delete sekmelerinde belirtilerek işlem tamamlanabilir.

Daha önceki bölümlerden hatırlanacağı üzere, AraKisi isimli depolanmış prosedür ad ve soyadı parametrelerini kullanmaktadır. Belirtilen SQL cümlesi veya depolanmış prosedür parametreleri sihirbaz tarafından veritabanından alınmakta ve aşağıda gösterilen parametre seçim sayfasında görüntülenmektedir.

Ekran Çıktısı 198, SQL Parametre Değerlerinin Belirtilmesi

Ekran çıktısında, Ad isimli parametre değerine sayfa üzerinde bulunan ctlAd isimli metin girdi kontrolü Text özelliğinin atanması belirtilmiştir. Parametre değerlerinin sayfa üzerinde bulunan kontroller dışında diğer yöntemlerle belirtimi desteklenmektedir.

Parametre değerlerinin alınması için kullanılabilecek yöntem ve açıklamalar aşağıda listelenmiştir.

| Yöntem | Açıklama |
| --- | --- |
| None | Parametre değerlerinin programcı tarafından belirleneceğini ifade etmektedir. |
| Cookie | Parametre değerlerinin çerez (ing. Cookie) değerlerinden alınması gerektiğini belirtmektedir. |
| Control | Parametre değerlerinin web sayfasında bulunan sunucu kontrolünün özellik değerinden alınması gerektiğini belirtmektedir. |
| Form | Parametre değerlerinin sayfa içerisinde bulunan form alanlarından alınması gerektiğini belirtmektedir. |
| Profile | Parametre değerlerinin kullanıcı profilinden alınması gerektiğini belirtmektedir. |
| QueryString | Parametre değerlerinin sayfa adres parametrelerinden alanlarından alınması gerektiğini belirtmektedir. Örneğin veribaglama.aspx?Ad=M gibi. |
| Session | Parametre değerlerinin oturumdan alınması gerektiğini belirtmektedir. |

Tablo 162, SQL Parametre Değelerleri

Aşağıdaki ekran çıktısında kontrol parametreleri (ing. Control Parametrers) özellikleri gösterilmiştir.

Ekran Çıktısı 199, Kontrol Parametresi Özellikleri

ConvertEmptyStringToNull, boş karakter dizi değerlerin ("") nasıl ele alınacağını belirtmektedir. Bu özelliğe True değer atanması durumunda, kontrol özellik değeri (örneğin ctlAd.Text özelliği) boş karakter dizi ise DefaultValue özelliğine atanan değer kullanılmaktadır.

Sihirbazın tamamlanmasıyla SQL veri kaynağı özelliklerine belirtilen değerler atanmış olacaktır. Son adımda, veri tablosu kontrolüne veri kaynağı olarak yapılandırılan SQL veri kaynağı kontrolünün atanması gerekmektedir.

Ekran Çıktısı 200, Veri Tablosuna Veri Kaynağının Atanması

Sayfanın internet tarayıcısı ile çalıştırılması sonucu aşağıdaki işlemler yapılmaktadır.

1. Sayfanın görüntülenmesi sonucunda http GET operasyonu yapılmaktadır. GET operasyonunun yapılması ile SqlDataSource1 nesnesi veritabanı bağlantısını yaparak veriye erişmektedir. Veriye erişimin tamamlanması ile IDataSource.DataSourceChanged olayı tetiklenmektedir.
2. Olayın tetiklenmesi ile veri tablosu kontrolüne veri bağlama işlemi yapılmaktadır.

Filtrele komutunun verilmesiyle aşağıdaki işlemler yapılmaktadır.

1. http POST işleminin yapılması sonucu veritabanı erişimi yapılmamakta, veri ViewState adı verilen form alan değerinden elde edilmektedir.
2. Sayfa üzerinde bulunan ve depolanmış prosedür parametreleriyle ilişkilendirilmiş olan kontrollerin Text özelliğinin değişmesi, POST işlemi yapılıyor olmasına rağmen, veritabanına tekrar erişim yapılmasını gerektirmektedir.

Aşağıdaki ekran çıktısında sayfa görünümü gösterilmiştir.

Ekran Çıktısı 201, SQL Veri Kaynağı ve Veri Bağlama Örneği

SqlDataSource olayları ve açıklamaları aşağıda listelenmiştir.

| Olay Adı | Açıklama |
| --- | --- |
| Deleted | Veri silme işlemi tamamlandığında tetiklenir. |
| Deleting | Veri silme işlemi başlamadan önce tetiklenir. |
| Filtering | Filtreleme işlemi başlamadan önce tetiklenir. |
| Inserted | Veri ekleme işlemi tamamlandığında tetiklenir. |
| Inserting | Veri ekleme işlemi başlamadan önce tetiklenir. |
| Selected | Veri seçme işlemi tamamlandığında tetiklenir. |
| Selecting | Veri seçme işlemi başlamadan önce tetiklenir. |
| Updated | Veri güncelleme işlemi tamamlandığında tetiklenir. |
| Updating | Veri güncelleme işlemi başlamadan önce tetiklenir. |

Tablo 163, System.Web.UI.WebControls.SqlDataSource Sınıfı Olay Bilgileri

Aşağıdaki örnekte Selecting olayı kullanılarak istenilen parametre değerinin belirtilmesi gösterilmiştir.

C#

**protected** **void** SqlDataSource1\_Selecting(**object** sender,   
 SqlDataSourceSelectingEventArgs e) {

DbCommand komut = e.Command;

komut.Parameters["@Ad"].Value = "xxx";

komut.Parameters["@Soyad"].Value = "yyy";

}

VB.NET

**Protected** **Sub** SqlDataSource1\_Selecting(**ByVal** sender **As** System.**Object**, \_  
 **ByVal** e **As** SqlDataSourceSelectingEventArgs) \_  
 **Handles** SqlDataSource1.Selecting

**Dim** komut **As** DbCommand = e.Command

komut.Parameters("@Ad").Value = "xxx"

komut.Parameters("@Soyad").Value = "yyy"

**End** **Sub**

Kod 423, SqlDataSource.Selecting Olayı Kullanımı

Benzer şekilde, örneğin kullanıcıdan alınmayan alan değerleri Inserting olayı ile belirlenebilir.

Nesne Veri Kaynağı Kontrolü

System.Web.UI.WebControls.ObjectDataSource sınıfı, verinin yansıma yardımıyla istenen nesneden alınabilmesini sağlamaktadır. Böylece, aşağıda şematik olarak gösterilen mimari web uygulamalarında kolaylıkla uygulanabilmektedir.

Şekil 62, Nesne Veri Kaynağı ve Çok Katmanlı Mimari

Nesne veri kaynağı kullanımını aktarmak amacıyla aşağıda gösterilen iş sınıfı kullanılacaktır.

C#

[DataObjectAttribute(true)]

**public** **class** KisiIsSinifi {

[DataObjectMethodAttribute(DataObjectMethodType.Select, true)]

**public** KisiDs.KisiDataTable KisiAra(**string** ad, **string** soyad) {

KisiTableAdapter adapter = **new** KisiTableAdapter();

**return** adapter.GetirAdSoyadIle(ad, soyad);

}

[DataObjectMethodAttribute(DataObjectMethodType.Insert, true)]

**public** **int** EkleKisi(**ref** **int** Id, **long** TcKimlikNo, **string** Ad, **string** Soyad,   
 DateTime DogumTarih) {

KisiTableAdapter adapter = **new** KisiTableAdapter();

**return** adapter.Insert(**ref** Id, TcKimlikNo, Ad, Soyad, DogumTarih);

}

[DataObjectMethodAttribute(DataObjectMethodType.Update, true)]

**public** **int** GuncelleKisi(**int** Id, **long** TcKimlikNo, **string** Ad, **string** Soyad,   
 DateTime DogumTarih) {

KisiTableAdapter adapter = **new** KisiTableAdapter();

**return** adapter.Update(Id, TcKimlikNo, Ad, Soyad, DogumTarih);

}

[DataObjectMethodAttribute(DataObjectMethodType.Delete, true)]

**public** **int** SilKisi(**int** Id) {

KisiTableAdapter adapter = **new** KisiTableAdapter();

**return** adapter.Delete(Id);

}

}

VB.NET

<DataObjectAttribute(True)> \_

**Public** **Class** KisiIsSinifi

<DataObjectMethodAttribute(DataObjectMethodType.Select, True)> \_

**Public** **Function** KisiAra(**ByVal** ad **As** **String**, **ByVal** soyad **As** **String**) \_  
 **As** KisiDs.KisiDataTable

**Dim** adapter **As** KisiTableAdapter = **New** KisiTableAdapter()

**Return** adapter.GetirAdSoyadIle(ad, soyad)

**End** **Function**

<DataObjectMethodAttribute(DataObjectMethodType.Insert, True)> \_

**Public** **Function** EkleKisi(**ByRef** Id **As** **Integer**, **ByVal** TcKimlikNo **As** **Long**, \_  
 **ByVal** Ad **As** **String**, **ByVal** Soyad **As** **String**, \_  
 **ByVal** DogumTarih **As** DateTime) **As** **Integer**

**Dim** adapter **As** KisiTableAdapter = **New** KisiTableAdapter()

**Return** adapter.Insert(Id, TcKimlikNo, Ad, Soyad, DogumTarih)

**End** **Function**

<DataObjectMethodAttribute(DataObjectMethodType.Update, True)> \_

**Public** **Function** GuncelleKisi(**ByVal** Id **As** **Integer**, **ByVal** TcKimlikNo **As** **Long**,\_  
 **ByVal** Ad **As** **String**, **ByVal** Soyad **As** **String**, \_  
 **ByVal** DogumTarih **As** DateTime) **As** **Integer**

**Dim** adapter **As** KisiTableAdapter = **New** KisiTableAdapter()

**Return** adapter.Update(Id, TcKimlikNo, Ad, Soyad, DogumTarih)

**End** **Function**

<DataObjectMethodAttribute(DataObjectMethodType.Delete, True)> \_

**Public** **Function** SilKisi(**ByVal** Id **As** **Integer**) **As** **Integer**

**Dim** adapter **As** KisiTableAdapter = **New** KisiTableAdapter()

**Return** adapter.Delete(Id)

**End** **Function**

**End** **Class**

Kod 424, Örnek İş Sınıfı

Oluşturulan sınıf Model projesi içerisinde bulunmaktadır.

Ekran Çıktısı 202, VS.NET Çözüm Penceresi ve İş Sınıfı

Kod parçasından görüleceği üzere, iş sınıfı metotları KisiTableAdapter metotlarını çağırmaktadır. İş sınıfı metotları içerisinde kullanıcı yetkilerinin kontrolü ve istisna yönetimi gibi işlemler yapılabilmektedir. Böylece, ara yüz katmanı sadece verinin kullanıcıya sunulması amacıyla kullanılabilmektedir.

System.ComponentModel.DataObjectAttribute öz niteliği, sınıfın nesne veri kaynağı tarafından kullanılabilir olduğunu belirtmektedir. System.ComponentModel.DataObjectMethodTypeAttribute özniteliği, donatıldığı metodun veri seçme, ekleme, güncelleme veya silme işlemlerinde kullanıldığını ifade etmektedir. Sınıf veya metotların nesne veri kaynağı ile kullanılabilmeleri için öznitelikler ile donatılması zorunlu değildir.

Aşağıdaki ekran çıktısında, ad ve soyadı alanlarından arama yapabilmek amacıyla metin girdi kontrolleri, verinin listelenmesi için veri tablosu kontrolü ve nesne veri kaynağı kontrolü tasarım zamanında gösterilmiştir.

Ekran Çıktısı 203, Nesne Veri Kaynağı Kontrolü

Configure Data Source … komutunun verilmesiyle sihirbaz başlatılmaktadır.

Ekran Çıktısı 204, Veri Kaynağı Tarafından Kullanılması İstenen Sınıf Seçimi

Listede sadece DataObjectAttribute özniteliği ile donatılmış olan sınıflar görüntülenmektedir. Buna rağmen Show only data components işareti kaldırılarak istenilen sınıf belirtilebilmektedir.

Ekran Çıktısı 205, Tüm Sınıfların Listelenmesi

Bir sonraki sihirbaz sayfasında verinin seçilmesi, güncellenmesi, eklenmesi ve silinmesi için kullanılacak iş sınıfı metotları seçilmektedir.

Ekran Çıktısı 206, İş Sınıf Metotları Seçimi

Verinin seçilmesi için kullanılan KisiIsSinifi.KisiAra isimli metot, ad ve soyadı parametrelerini almaktadır. Alınan parametre değerlerinin nasıl belirleneceği bir sonraki sihirbaz sayfasında gösterilmektedir.

Ekran Çıktısı 207, Parametre Kaynaklarının Belirlenmesi

Son olarak veri tablosu kontrolünün veri kaynağına bağlanması aşağıda gösterilmiştir.

Ekran Çıktısı 208, Veri Tablo Kontrolünün Veri Kaynağına Bağlanması

Nesne veri kaynağı, veri güncelleme ve silme işlemlerinde verinin orijinal anahtar değerini de metotlara iletmektedir. Buna rağmen, her tabloda, değeri kullanıcı tarafından değiştirilemeyen otomatik artan sayı alanının ana anahtar olarak kullanımının benimsenmesi, eski anahtar değerlerin kullanımını gerektirmemektedir.

ObjectDataSource.OldValuesParameterFormatString özelliği, eski değerleri metotlara iletmek için kullanılan parametre format değerini içermektedir. Sınıf tanımından, metotların eski alan değerlerini kullanmadığı görülmektedir. Dolayısıyla bu değerin aşağıdaki şekilde düzenlenmesi gerekmektedir.

Ekran Çıktısı 209, Nesne Veri Kaynağı Özellikleri

Sayfanın internet tarayıcı ile görüntülenmesi aşağıda gösterilmiştir.

Ekran Çıktısı 210, Nesne Veri Kaynağı Kullanılarak Bağlanan Veriler

Nesne veri kaynağı, sunmuş olduğu kullanım kolaylıkları ve iş nesnelerini kullanabilmesi değerlendirildiğinde web uygulamalarında rahatlıkla tercih edilebilmektedir.

Uygulama yazılımlarında direkt ObjectDataSource kullanımı yerine, programcı tarafından bu sınıfı miras alarak oluşturulan kontrolün kullanımı önerilmektedir. Böylece, kontrol davranışı istenilen şekilde düzenlenebilmektedir.

ObjectDataSource olayları kullanımı hakkında bilgi almak için SqlDataSource olay kullanım örneklerini inceleyiniz.

Programcı Tanımlı Veri Kaynak Kontrolleri

Programcılar tarafından veri kaynağı kontrolleri ve görünüm sınıfları oluşturularak, ASP.NET veri bağlama mimarisi tarafından sunulan tüm olanaklar kullanılabilmekte, tanımlanan sınıflar yardımıyla istenilen veri ara yüz kontrollerine iletilebilmektedir.

Kontrol Projesi Eklenmesi

Gerek veri kontrolleri gerek diğer programcı tanımlı ASP.NET kontrollerinin bulunduğu sınıfların ayrı bir Assembly dosyası içerisinde bulunması önerilmektedir.

Ekran Çıktısı 211, ASP.NET Kontroller Projesi Eklenmesi

Kontrol ve Görünüm Sınıfları Gerçekleştirimleri

Aşağıda tek görünüm içeren veri kaynak kontrolleri için şablon olarak kullanılabilecek veri kaynağı ve görünümü sınıfları gösterilmiştir.

C#

// Veri kaynağı kontrolü

[ToolboxData("<{0}:OrnekVeriKaynagi runat=server></{0}:OrnekVeriKaynagi>")]

**public** **class** OrnekVeriKaynagi : DataSourceControl {

**private** OrnekVeriKaynakGorunumu gorunum = **null**;

**public** OrnekVeriKaynagi()

{

}

**protected** **override** DataSourceView GetView(**string** viewName) {

**if** (gorunum == **null**)

gorunum = **new** OrnekVeriKaynakGorunumu(**this**, "GorunumAd");

**return** gorunum;

}

**protected** **override** ICollection GetViewNames() {

**return** **new** **string**[] { "GorunumAd" };

}

}

// Veri kaynak görünümü

**class** OrnekVeriKaynakGorunumu : DataSourceView {

**internal** OrnekVeriKaynakGorunumu(IDataSource kaynak, **string** ad)

: **base**(kaynak, ad) {

}

**protected** **override** **int** ExecuteDelete(IDictionary keys,   
 IDictionary oldValues) {

// Silme işleminin yapıldığı kod bölümü.

}

**protected** **override** IEnumerable ExecuteSelect(  
 DataSourceSelectArguments arguments) {

// Veri seçme işleminin yapıldığı kod bölümü.

}

**protected** **override** **int** ExecuteInsert(IDictionary values) {

// Veri ekleme işleminin yapıldığı kod bölümü.

}

**protected** **override** **int** ExecuteUpdate(IDictionary keys, IDictionary values,  
 IDictionary oldValues) {

// Veri güncelleme işleminin yapıldığı kod bölümü.

}

}

VB.NET

' Veri kaynağı kontrolü

<ToolboxData("<{0}:OrnekVeriKaynagi runat=server></{0}:OrnekVeriKaynagi>")> \_

**Public** **Class** OrnekVeriKaynagi

**Inherits** DataSourceControl

**Private** gorunum **As** OrnekVeriKaynakGorunumu = **Nothing**

**Public** **Sub** **New**()

**End** **Sub**

**Protected** **Overrides** **Function** GetView(**ByVal** viewName **As** **String**) \_  
 **As** DataSourceView

**If** gorunum **Is** **Nothing** **Then**

gorunum = **New** OrnekVeriKaynakGorunumu(**Me**, "GorunumAd")

**End** **If**

**Return** gorunum

**End** **Function**

**End** **Class**

' Veri kaynak görünümü

**Class** OrnekVeriKaynakGorunumu

**Inherits** DataSourceView

**Friend** **Sub** **New**(**ByVal** kaynak **As** IDataSource, **ByVal** ad **As** **String**)

**MyBase**.**New**(kaynak, ad)

**End** **Sub**

**Protected** **Overrides** **Function** ExecuteDelete(**ByVal** keys **As** IDictionary, \_  
 **ByVal** oldValues **As** IDictionary) **As** **Integer**

' Silme işleminin yapıldığı kod bölümü.

**End** **Function**

**Protected** **Overrides** **Function** ExecuteSelect(**ByVal** arguments \_  
 **As** DataSourceSelectArguments) **As** IEnumerable

' Veri seçme işleminin yapıldığı kod bölümü.

**End** **Function**

**Protected** **Overrides** **Function** ExecuteInsert(**ByVal** values **As** IDictionary) \_  
 **As** **Integer**

' Veri ekleme işleminin yapıldığı kod bölümü.

**End** **Function**

**Protected** **Overrides** **Function** ExecuteUpdate(**ByVal** keys **As** IDictionary, \_  
 **ByVal** values **As** IDictionary, \_  
 **ByVal** oldValues **As** IDictionary) **As** **Integer**

' Veri güncelleme işleminin yapıldığı kod bölümü.

**End** **Function**

**End** **Class**

Kod 425, Şablon Veri Kaynak ve Görünüm Sınıfları

Veri Kontrolleri Kullanımı

Verinin gösterimi ve düzenlenmesi için kullanılabilecek ASP.NET sunucu kontrolleri ve açıklamaları aşağıda gösterilmiştir.

| Kontrol | Açıklama |
| --- | --- |
| GridView | Verinin tablo şeklinde görüntülendiği ve kullanıcı tarafından düzenlemelerin yapılabileceği kontroldür. |
| DataList | Veri tablo kontrolünden daha az işleve sahip liste kontrolüdür. |
| DetailsView | Veri alanlarının ad – değer şeklinde görüntülenebileceği ve düzenlenebileceği kontroldür. |
| FormView | Verinin istenen şekilde görüntülenebilmesi ve düzenlenebilmesini sağlayan kontroldür. |
| Repeater | Şablonlar (ing. Templates) yardımıyla verinin programcı tarafından listelenmesini sağlayan kontroldür. |
| ReportViewer | Veri raporlama amacıyla kullanılan kontroldür. |

Tablo 164, ASP.NET Ara Yüz Kontrolleri ve Açıklamaları

Ara yüz kontrolleri ve kontrollerin sağlamış oldukları işlevler değişmekle birlikte, verinin bağlanması ve temel kullanım özellikleri benzerlik göstermektedir.

System.Web.UI.WebControls.CompositeDataBoundControl soyut sınıfı, tablo verilerinin bütün olarak ele alındığı veri kontrolleri için ata sınıftır. GridView, DetailsView ve FormView kontrolleri bu sınıfı miras almaktadır.

ASP.NET sunucu kontrolleri detayları oldukça geniş ve detaylı bir konudur. Aşağıdaki konu başlıklarında temel kullanım özellikleri hakkında bilgiler verilecektir.

Verinin Bağlanması

Tasarım anında verinin bağlanması için VS.NET özellikler penceresi ile DataSourceID özelliğinin veya kontrol üzerindeki akıllı menünün (ing. Smart Menu) kullanılması gerekmektedir.

Aşağıdaki ekran çıktısında DetailsView kontrolüne verinin bağlanması gösterilmiştir.

Ekran Çıktısı 212, DetailsView Kontrolü ve Verinin Bağlanması

Veri Alanlarıyla Çalışmak

GridView ve DetailsView kontrolleri, verinin bağlanmasıyla otomatik olarak alan nesnelerini oluşturmaktadırlar. Oluşturulan alan nesneleri akıllı menü içerisinde bulunan Edit Fields … komutu ile düzenlenebilmektedir.

Ekran Çıktısı 213, Alan Editörü

Auto-generate fields seçeneği, alan nesnelerinin çalışma zamanında oluşturulmasını sağlamaktadır. Editör yardımıyla yeni alanlar eklenmesi veya mevcut alan bilgilerinin görsel olarak düzenlenmesi mümkündür. Convert this field into a TemplateField komutu ile, içeriği programcı tarafından tasarlanabilen şablon alan oluşturulmaktadır.

Ekran Çıktısı 214, Yeni Alan Eklenmesi

Yukarıdaki ekran çıktısında, bağlantı oluşturmak için kullanılan HyperLinkField tipinde yeni bir alan eklenmesi gösterilmiştir. Bağlantıya tıklandığında, kişi kayıt numarası parametre geçilerek adresler.aspx isimli sayfa internet tarayıcısı tarafından çağrılmaktadır.

Ekran Çıktısı 215, Yeni Eklenen Bağlantı Alanı Görünümü

Şablonlar

Şablonlar (ing. Templates), kontrollerin belirli bölümlerinin istenildiği şekilde tasarlanabilmesini sağlamaktadırlar.

Ekran Çıktısı 216, Şablon Alan Ekleme

Yukarıdaki ekran çıktısında, DetailsView kontrolüne yeni bir şablon alan eklenmiştir. Edit Templates komutu verilerek eklenen şablon alanın tasarımının yapılabilmesi için açılan şablon editörü aşağıda gösterilmiştir.

Ekran Çıktısı 217, Şablon Editörü

Örneğin, şablon alana ASP.NET takvim kontrolü sürüklenerek kişi doğum tarihinin takvim kontrolü ile görüntülenmesi sağlanabilir.

Ekran Çıktısı 218, Şablon Alan Kullanımı

Edit Data Bindings… komutu verilerek, şablon içerisine eklenen kontrolün hangi alan ile ilişkilendirileceği tanımlanabilir.

Ekran Çıktısı 219, Kontrol Özelliğinin Alana Bağlanması

Ekran çıktısında, Calendar.SelectedDate özelliğinin çift yönlü olarak DogumTarih alanına bağlanması gösterilmektedir.

Format değerine, kitabın Sık Kullanılan Sınıflar ve Öneriler bölümlerinde aktarılan veri format değerleri atanabilmektedir.

Veri Sayfalama

Web uygulamalarında sık yapılan ve sistem performansına önemli etkisi olan konulardan birisi de verinin kullanıcıya sayfalanarak gösterilmesidir.

Konuya başlamadan önce kitabın Veritabanı Kullanım Yöntemleri bölümü Veri Sayfalama konu başlığının gözden geçirilmesi önerilmektedir.

İstemci Taraflı Veri Sayfalama

İstemci taraflı veri sayfalama, veri kaynağı tarafından sağlanan bütün verinin, örneğin GridView veya FormView kontrolü ile, ara yüz kontrolü tarafından kullanıcıya sayfalar şeklinde gösterimini ifade etmektedir. Bu yöntemde, veri kaynağı tüm veriyi sağlamakta, yapılan işlem sadece verinin sayfalar şeklinde gösterimi olmaktadır.

Az sayıda veri için tercih edilebilecek bu yöntem, kontrollerin EnablePaging özelliklerine True değerinin atanması ile kolaylıkla yapılabilmektedir.

Örneğin GridView.AllowPaging özelliğine True, GridView.PageSize özelliğine istenilen sayfa boyutu değerinin atanmasıyla veri tablo kontrolü otomatik olarak sayfalama yapmaktadır.

Ekran Çıktısı 220, İstemci Taraflı Veri Sayfalama

Sunucu Taraflı Veri Sayfalama

Sunucu taraflı veri sayfalama, veri kaynağından sadece gösterilmek istenen sayfa verisinin alınmasını ifade etmektedir. Dolayısıyla sayfalama işlevi, ara yüz kontrolü yenine veri kaynağı kontrolü tarafından yapılmaktadır. Bu konu başlığında ObjectDataSource kontrolü kullanılarak sunucu taraflı veri sayfalamanın SQL Server veritabanı ile yapılması gösterilecektir. Aynı işlemin SqlDataSource kontrolü ile de kolaylıkla yapılması mümkündür.

**Depolanmış Prosedürler**

Verinin sayfalı olarak alınabilmesi için oluşturulan depolanmış prosedürler aşağıda gösterilmiştir.

**Create** **Procedure** [AraKisiSayfalayarak]

(

@Ad nvarchar(25) = **Null**, @Soyad nvarchar(25) = **Null**,

@Siralama nvarchar(25) = "Ad", @SatirBaslangic int,

@SayfaBoyut int

)

**AS**

**with** Liste **as** (**select** Row\_Number() over (

**Order** **By** **Case** @Siralama

**When** 'Ad' **Then** Ad

**End**,

**Case** @Siralama

**When** 'Ad DESC' **Then** Ad

**End** **Desc**,

**Case** @Siralama

**When** 'Soyad' **Then** Soyad

**End**,

**Case** @Siralama

**When** 'Soyad DESC' **Then** Soyad

**End** **Desc**

)

**AS** **Row**, Id, TcKimlikNo, Ad, Soyad, DogumTarih

**from** Kisi **where** (@Ad **is** **Null** **OR** Ad **Like** @Ad +'%') **And**

(@Soyad **is** **Null** **Or** Soyad **Like** @Soyad +'%')

)

**Select** Id, TcKimlikNo, Ad, Soyad, DogumTarih **from** Liste

**Where** **Row** **between** @SatirBaslangic+1 **and** @SatirBaslangic+@SayfaBoyut+1;

Kod 426, Kişi Tablosu Verisine Sayfalayarak Erişim

**Create** **Procedure** [GetirKisiSayisi]

(

@Ad nvarchar(25), @Soyad nvarchar(25)

)

**As**

**Begin**

**Select** **count**(\*) **From** [Kisi]

**Where** (@Ad **is** **Null** **OR** Ad **Like** @Ad +'%') **And**

(@Soyad **is** **Null** **Or** Soyad **Like** @Soyad +'%')

**END**

Kod 427, Toplam Kayıt Sayısına Erişim

Hazırlanan AraKisiSayfalayarak isimli depolanmış prosedür, sıralama desteği ile birlikte, ad ve soyadına göre arama yapmakta ve parametre olarak iletilen sayfada bulunan kişi kayıtlarını döndürmektedir.

GetisKisiSayisi isimli depolanmış prosedür, parametre olarak iletilen ad ve soyadı değerine uyan kayıt sayısını döndürmektedir. Böylece, toplamda kaç sayfa bulunacağı bu prosedür yardımıyla elde edilebilmektedir.

**Veri Tablo Adaptörünün Düzenlenmesi**

Oluşturulan depolanmış prosedürlere erişim için kişi tablo adaptörüne eklenen yeni sorgular aşağıda gösterilmiştir.

Ekran Çıktısı 221, Tablo Adaptörü ve Sayfalama Sorguları

**İş Sınıfı Metotları**

KisiIsSinifi isimli iş sınıfına sayfalama yapabilmek ve toplam kayıt sayısını alabilmek için aşağıda gösterilen metotlar eklenmelidir.

C#

[DataObjectMethodAttribute(DataObjectMethodType.Select)]

**public** KisiDs.KisiDataTable KisiAra(**string** ad, **string** soyad, **string** siralama,  
 **int** satirBaslangic, **int** sayfaBoyut) {

KisiTableAdapter adapter = **new** KisiTableAdapter();

**return** adapter.GetirAdSoyadIleSayfali(ad, soyad, siralama, satirBaslangic,   
 sayfaBoyut);

}

**public** **int** GetirKisiSayisi(**string** ad, **string** soyad, **string** siralama,  
 **int** satirBaslangic, **int** sayfaBoyut) {

KisiTableAdapter adapter = **new** KisiTableAdapter();

**return** (**int**)adapter.GetirKisiSayisiAdSoyadIle(ad, soyad);

}

VB.NET

<DataObjectMethodAttribute(DataObjectMethodType.**Select**)> \_

**Public** **Function** KisiAra(**ByVal** ad **As** **String**, \_  
 **ByVal** soyad **As** **String**, \_  
 **ByVal** siralama **As** **String**, \_  
 **ByVal** satirBaslangic **As** **Integer**, \_  
 **ByVal** sayfaBoyut **As** **Integer**) **As** KisiDs.KisiDataTable

**Dim** adapter **As** KisiTableAdapter = **New** KisiTableAdapter()

**Return** adapter.GetirAdSoyadIleSayfali(ad, soyad, siralama, \_  
 satirBaslangic, sayfaBoyut)

**End** **Function**

**Public** **Function** GetirKisiSayisi(**ByVal** ad **As** **String**, **ByVal** soyad **As** **String**, \_  
 **ByVal** siralama **As** **String**, \_  
 **ByVal** satirBaslangic **As** **Integer**, \_  
 **ByVal** sayfaBoyut **As** **Integer**) **As** **Integer**

**Dim** adapter **As** KisiTableAdapter = **New** KisiTableAdapter()

**Return** **CType**(adapter.GetirKisiSayisiAdSoyadIle(ad, soyad), **Integer**)

**End** **Function**

Kod 428, İş Sınıfı Sayfalama Metotları Eklenmesi

**Nesne Veri Kaynağı Düzenlemeleri**

Veriye erişim için nesne veri kaynağının iş sınıfının yüklenmiş KisiAra metodunu kullanması aşağıda gösterilmiştir.

Ekran Çıktısı 222, Yüklenmiş KisiAra Metodunun Seçimi

Veri sayfalama, aynı zamanda sunucu tarafında sıralama amacıyla, nesne veri kümesi özellikleri aşağıdaki şekilde düzenlenmelidir.

| Özellik | Açıklama |
| --- | --- |
| MaximumRowsParameterName | Sayfa boyutunu iş nesnesine iletmek için kullanılan parametre adını belirtmektedir. Örnekte bu değer, KisiAra isimli metodun sayfaBoyut parametresi olarak düzenlenmelidir. |
| SelectCountMethod | Toplam kayıt sayısını elde etmek için kullanılması gereken iş sınıfı metodunu belirtmektedir. Örnekte bu değer GetirKisiSayisi olarak düzenlenmelidir. |
| SortParameterName | Veri sıralamasında kullanılmak istenen parametre adını belirtmektedir. Örnekte bu değer, siralama olarak düzenlenmelidir. |
| StartRowIndexParameterName | Sayfalamanın başlayacağı satır numarasının parametre adını belirtmektedir. Örnekte bu değer satirBaslangic olarak düzenlenmelidir. |

Tablo 165, Nesne Veri Kaynağı Sayfalama ve Sıralama Özellikleri

Nesne veri kaynağı özellikleri tasarım anında aşağıda gösterilmiştir.

Ekran Çıktısı 223, Nesne Kaynağı Özellikleri

**Veri Tablosu Kontrolünün Düzenlenmesi**

Sunucu taraflı veri sayfalama için, istemci taraflı veri sayfalamada olduğu gibi, sadece GridView.AllowPaging ve GridView.PageSize özelliklerinin düzenlenmesi yeterlidir.

Sıralama desteği için GridView.AllowSorting özelliğine True değerinin atanması gerekmektedir.

Ekran Çıktısı 224, Veri Tablosu Kontrolü Sıralama Desteği

Sıralama, kullanıcının veri tablosu kontrolü alan isimlerine tıklaması ile artan ve azalan şekilde yapılmaktadır.

Sıralamanın yapıldığı tablo alanları üzerinde indekslerin bulunması önem taşımaktadır. Veri tablosu kontrolünde sadece ad ve soyadı alanlarında sıralama yapılabilmesi için alan editörü yardımıyla diğer alanların SortExpression özelliğine boş değer atanması yeterlidir.

Ekran Çıktısı 225, Sıralama Desteğinin Alan Bazında Düzenlenmesi

Yapılan işlemler sonucu kayıtların sayfalanması ve sıralanması tamamen sunucu tarafında yapılmaktadır.

İşlem sonucunda elde edilen çeşitli ekran çıktıları aşağıda gösterilmiştir.

Ekran Çıktısı 226, Ad Alanı Bazında Artan Sıralama

Ekran Çıktısı 227, Ad Alanı Bazında Azalan Sıralama

Ekran Çıktısı 228, Filtrelenmiş 2. Sayfa Görünümü

Ekran Çıktısı 229, Filtrelenmiş ve Soyadı Sıralanmış Görünüm

PDA Uygulamaları

PDA (Personal Digital Asistant) olarak adlandırılabilen akıllı aygıtlar üzerinde .NET Compact Framework üzerinde çalışan uygulamaların veri bağlama mimarisi Windows Forms.NET uygulamaları ile benzerlik göstermektedir. Buna rağmen, cihaz kapasiteleri değerlendirilerek Windows Forms.NET mimarisinde sunulan bazı işlevler PDA uygulamalarında kullanılamamaktadır.

Veri kaynağı penceresi ile istenen verinin form üzerine sürüklenmesi ile veri bağlama işlemi kolaylıkla yapılabilmektedir.

Ekran Çıktısı 230, PDA Uygulamaları Veri Bağlama Örneği

Ekran çıktısında, veri bağlamanın web hizmeti ile yapılması gösterilmiştir.

Veri Bağla komutunu verildiğinde çalıştırılan kod aşağıda gösterilmiştir.

C#

Servis hizmet = **new** Servis();

KisiDs.KisiDataTable tablo = hizmet.GetirAdSoyadIle(**null**, **null**);

kisiBindingSource.DataSource = tablo;

VB.NET

**Dim** hizmet **As** Servis = **New** Servis()

**Dim** tablo **As** KisiDs.KisiDataTable = hizmet.GetirAdSoyadIle(**Nothing**, **Nothing**)

kisiBindingSource.DataSource = tablo

Kod 429, BindingSource ve PDA Uygulamaları

İndeks

A

Activator 208, 396

Active Server Pages 822

ActiveX Database Objects 822

Administrator Policy 340

ADO.NET 822, 825, 833

AES 692

Anonymous Method 304

ANSI 489

API 14

App.Config 334

AppDomain 384, 738

Application Domains 382

Application Settings Section 762

ApplicationException 254

AppSettingsSection 768

Array 79, 82

ArrayList 445, 452, 454

ASCII 488, 491

ASCIIEncoding 490

aspnet\_regiis 795

Assembly 60, 315, 366

Assembly Binding Log Viewer 347

Assembly Linker 357

Assembly Manifest 332

AssemblyFileVersion 344

AssemblyInformationalVersion 344

AssemblyKeyFile 332

AssemblyVersion 344

Asymmetric Encryption 692

AsymmetricAlgorithm 715

Asynchronous Programming Model 613

Attributes 389, 397

AttributeTargets 403

Authentication 735

AuthenticationException 748

Authorization 735

AverageTimer32 673

B

Base64 645

Bilgisayar 10

BinaryFormatter 540

BinaryReader 487, 519, 520

BinaryWriter 487, 519, 520

Binding Policy 335

BindingList 974

BindingNavigator 990

BindingSource 989, 997

Bit 10

BitArray 445, 458

Bitwise Operations 92

BNF 69

Boolean 65

Borland Database Engine 821

Borland Developer Studio 21

Boxing 153

Branching 70

BufferedStream 487, 533

Build Action 364

Business Entities 818

Business Intelligence 37

Business Layer 817

Business Objects 818

Byte 64

C

Certificate Authority 700

Certificate Store 727

Char 65

Charset 489

Chipher Text 689

CIL 16

Class 60, 127

Client – Server 827

Client-Side Paging 881

CLS 63

CLSCompliant 406

Code Access Security 749

Code Behind 943

Collection<T> 478

CollectionBase 445

CommandBehavior 854

Comment 73

Compile 11

CompositeDataBoundControl 1025

configProtectedData 763

configProtectionProvider 795

configSections 763

Configuration 764, 766

Configuration Property 762

Configuration Section 761

Configuration Section Group 761

ConfigurationElement 768

ConfigurationManager 764, 765

ConfigurationProperty 768, 773

ConfigurationPropertyAttribute 773

ConfigurationPropertyCollection 773

ConfigurationSection 767

ConfigurationSectionGroup 768, 777

Connection Pool 827

connectionStrings 762

ConnectionStringsSection 768

Constraint 862

Constructor 154

Convert 642

CounterCreationData 675

CounterDelta32 673

CPU 10

Cryptanalysis 689

Cryptographic Service Provider 702

Cryptography 689

Cryptology 689

CspParameters 704

CTS 17, 63

Culture 84

Cursor 850

D

Data Access Objects 818

Data Binding 935

Data Compression 533

Data Conversion 642

Data Persistency Layer 857

Data Protection API 707

Data Provider 823

Data Signature 698

Data Source 982

DataAccess Layer 817

Database Connection String 834

Database View 875

DataColum 878

DataColumn 862, 865

DataGridView 990

DataList 1024

DataObjectAttribute 1016

DataObjectMethodTypeAttribute 1016

DataRow 862, 866, 876

DataSet 829, 862, 965

DataSourceControl 1003

DataSourceView 1004

DataTable 864, 873, 876, 877, 964

DataView 862, 868

DateTime 626

DbCommand 825

DbConnection 825, 833

DbDataAdapter 826, 858, 869, 873

DbDataReader 825, 850

DbParameter 826, 846

DbParameterCollection 846

DbTransaction 826, 892

Decimal 65

Decryption 689

Decryption Key 690

Deflate 534

DeflateStream 487

Delayed Signing 331

Delegate 288

Deployment 947

DES 692, 709

Design Pattern 935

Destructor 232

DetailsView 1024

Dictionary<T> 477

DictionaryBase 445

Digital Certificate 699

Directory 486

DirectoryInfo 486, 496, 499

Dirty Read 890

Disconnected Model 826

Disposable Object 232

Distributed Transaction 891

DllImport 406, 407

Double 64

DriveInfo 486, 501

DSACryptoServiceProvider 724, 725

E

Early Binding 396

ElapsedTime 673

Encapsulation 126, 170

Encoding 488, 490

Encryption 689

Encryption Key 690

Entroph 708

Enum 95

Enumarated Types 91

Environment 646

Event 306

Event Log 661

Event Source 662

Event Type 662

Event Viewer 662

EventArgs 300, 302

EventHandler 304

EventLog 663

EventWaitHandle 599

Exception 251, 264

Exception Filtering 276

Exception Publishing 252

Exception Throwing 259

Executable Code Generation 12

Explicit Type Casting 116

Expression 74

F

FIFO 472

File 486

FileAccess 527

FileInfo 486, 496, 497

FileMode 526

FileNotFoundException 253

FileShare 528

FileStream 487, 526

FileSystemInfo 486, 496

FileSystemWatcher 486, 506

Finalizable 230

Format String 633

FormatException 253, 642

Formatter 540

FormsIdentity 741

FormView 1024

Friend 173

G

gacutil.exe 323, 337

Garbage Collector 163, 227

Generic 213

GenericIdentity 741

GET 1012

GetType 390

Global Assembly Cache 321

Globally Unique Identifier 648

GridView 1007, 1024

Guid 648

GZip 534

GZipStream 487

H

Hash 247, 326, 695

HashAlgorithm 721

HashTable 445, 461

HMACSHA1 720

HybridDictionary 446, 471

I

IAsyncResult 615

IBindingList 963

IBindingListView 963

ICollection 431, 436

IComparable 449

IConvertible 643

ICryptoTransform 711

ICurrencyManagerProvider 989

IDataSource 1003, 1012

IDE 35

Identifier 70

IDeserializationCallback 551

IDictionary 431, 437, 441

IDisposable 230, 232

IEnumarable 431

IEnumerable 432

IEnumerator 432

IEqualityComparer 451

IIdentity 737

IL Kod 13

IList 431, 437, 963

IListSource 964, 1003

Imperative Role Based Security 753

Implementation of interface 202

Implicit Type Casting 116

Implicitly Typed Local Variables 78

Imports 172

Inheritance 126, 170

InnerException 266

Int16 64

Int32 64

Int64 64

Interface 130, 287

Interlocked 584

Intermediate Code Generation 12

Internet Information Server 941

InvalidCastException 253, 476

IPermission 749

IPrincipal 737

IResourceReader 351

IResourceWriter 351

ISerializable 549

Isolated Storage 702

IsolatedStorageFile 487

IsolatedStorageFileStream 487

ISupoortInitiliazeNotification 963

ISupportInitialize 963

ITypedList 963

IV 710

İ

İkil 10

internal 173

J

Java Virtual Machine 11

K

Key Database 703

Keyed Hash Algorithm 698

KeyedCollection<T> 478

L

Late Binding 396

LEX 69

Lexical Analysis 12

LIFO 473

Lightweight Directory Access Protocol 828

List<T> 477

ListDictionary 446, 471

Local Security Authority 708

Local Transaction 891

Localization 350

lock 588

M

Machine Code 10

Machine Policy 340

Main 376

makecert.exe 729

Managed Code 17

Managed Platform 375

MD5 696

MD5CryptoServiceProvider 720

Member Shadowing 188

MemoryStream 486, 530

Message Authentication Code 697

Method Overloading 145

Microsoft Certificate Services 701

Microsoft Management Console Application 662

Model View Controller 935

Model-View-Controller 982

Monitor 590

mscorsvr.dll 377

mscorwks.dll 377

MSDataSetGenerator 905

MSIL 16

MulticastDelegate 305

Multi-Tasking Operating System 557

Multi-Threaded Application 376, 555

Mutex 599

N

Namespace 60

NameValueCollection 446

NameValuePair<T> 478

nesne 127

Neutral Culture 348

Nonrepetable Read 890

NonSerializedAttribute 548

nothing 150

NotInheritable 187

null 150

Nullable 90

Nullable Types 89

NullReferenceException 253

NumerOfItems32 673

O

Object 63, 65, 74, 154, 230

Object Linking and Embedding Database 821

Object Oriented Programming 123

ObjectDataSource 1014

Obsolete 398, 407

OLEDB Provider 821

OnDeserializedAttribute 551

OnDeserializingAttribute 551

One-way Data Binding 962

OnSerializedAttribute 551

OnSerializingAttribute 551

Open Database Connectivity 821

OpenSSL 703

Operand 97

Operating System 10

Operatör 97

Optimization 12

OrderedDictionary 446, 469

OutOfMemoryException 253

Overridable 190

P

Parallel Computing 556

partial 199

PassportIdentity 741

Path 486, 502

PBKDF1 706

Performance Logs and Alerts 668

PerformanceCounter 678

PerformanceCounterCategory 675

PerformanceCounterCategoryType 677

Personal Digital Asistant 1037

Phantom Read 891

PKCS #12 729

Plain Text 689

Polymorphism 126, 170

Portable Executable 25, 315

POST 1012

Pre-Processor 118

Primary Thread 376, 555

Primitive Types 74

PrincipalPermission 753

PrincipalPermissionAttribute 751

PrincipalPolicy 739

Private Assembly 318

Private Key 326, 693

Process 373, 378

Protected Configuration Provider 793

ProtectedConfigurationProvider 796

ProtectedConfigurationSection 768

ProtectedMemory 709

Provider Design Pattern 788

ProviderBase 788

Public Key 326, 692

Public Key Infrastructure 326, 693

Publisher Policy Assembly 338

Q

QuadroSexagesimal Encoding 645

Queue 445, 472

Queue<T> 477

R

RaiseEvent 308

Random Number Generation 705

RateOfCountsPerSecond 673

RC2 709

Read Comitted 890

ReaderWriterLock 594

ReadOnlyCollection<T> 478

ReadOnlyCollectionBase 445

Record Fetch 850

Reflection 29, 209, 389

Registry 541, 653

RegistryHive 654

RegistryKey 654

RegistryValueKind 654

Repeatative Structures 70

Repeater 1024

Report Definition Language 999

ReportViewer 990, 999, 1024

Reserved Word 73

Resource File Generator 349

ResourceManager 354, 356, 360

ResourceReader 354

ResourceWriter 353

ResXResourceReader 353

ResXResourceWriter 351

Rethrowing Exception 261

Rfc2898DeriveBytes 706

RIPEMD160Managed 720

RISC 13

Rijndael 692

RijndaelManaged 709

RNGCryptoServiceProvider 705

Role Based Authorization 736

Root CA 701

Row Level Locking 889

RSACryptoServiceProvider 715, 724

RSAParameters 715

S

Salted Hash 697

Satellite Assembly 356

SByte 64

sc.exe 805

Scalibility 826

Scope 72

Sealed 187

sectionGroup 764

Secure Socket Layer 693

SecurityException 254, 752

Self Signed Certificate 700

Semantics Analysis 12

Semaphore 599

Sequential Implementation 70

Serializable 406, 539

SerialPort 487

Server Control 1003

Server-Side Paging 882

Service Account 803

Service Application 661

Service Control Manager 802

Service Dependency 804

Service Oriented Architecture 9

Service Process Installer 813

ServiceBase 805, 807

ServiceMain 376

Setting Provider 787

Settings File 779

SettingsProviderAttribute 788

SHA1 696

SHA1CryptoServiceProvider 720

Shared 159

Shared Assembly 320

Shared Resource 558

Sınıf 127

Single 64

Single-Threaded Application 376, 555

Smart Card 695

SoapFormatter 540

Software 9

Software Development Kit 22

Software Engineering 9

SortedList 445, 464

SortedList<T> 477

SpecialFolder 505

Specific Culture 348

SqlConnection 837

SqlDataSource 1006, 1013

SSL Handshake 695

Stack 445, 473

Stack<T> 477

StackTrace 263, 272

Statement 71

static 159

Stream 486, 509

StreamReader 487, 519, 523

StreamWriter 487, 519, 523

String 65, 84

StringCollection 445

StringDictionary 446

StringReader 487

StringWriter 487

Strong Named Assembly 324

Strong Naming 324

Structure 60

Subordinate CA 701

Symmetric Encryption 691

SymmetricAlgorithm 709

SyncLock 588

Syntax Analysis 12

System.Collections 431, 444, 935

System.Collections.Generic 443, 444, 475

System.Collections.ObjectModel 444, 475

System.Collections.Specialized 444

System.ComponentModel 935

System.Data.Odbc 823

System.Data.OleDb 823

System.Data.OracleClient 823

System.Data.SqlClient 823

System.IO 485

System.IO.Compression 485

System.IO.IsolatedStorage 485

System.IO.Ports 485

System.Reflection 389

System.Security.Cryptography.X509Certificates 730

SystemException 253

T

Table Adapter 904

Template 213, 1027

TextReader 487, 511

TextWriter 487, 511

Thread 555, 559

Thread Safe 559

ThreadAbortException 575

ThreadPool 604

ThreadPriority 562

ThreadState 563

Timer 609

TimeSpan 630

Transaction 819, 898

Transaction Isolation Level 890

Transaction Savepoint 894

TransactionScope 897

TripleDES 709

Trust Relationship 701

Two-way Data Binding 962

Type 389, 394

Type Casting 114, 642

Typed Dataset 903

typeof 390

U

UInt16 64

UInt32 64

UInt64 64

UnBoxing 153

Unicode 491

UnicodeEncoding 490

Unmanaged Code 17

Unmanaged Platform 375

Unsafe 114

Untyped Dataset 903

USB token 695

User Interface 817

Using 172, 234

UTF32 491

UTF32Encoding 490

UTF8 491

UTF8Encoding 490

V

Value Type 476

ValueType 227

virtual 190

Virtual Member Table 192

Visual Component Library 21

Visual Studio Development Server 940

Visual Studio.NET 35

Volatility 592

W

Web Method 943

Web Server 940

WebConfigurationManager 765

WebException 254

WebMethod 406

Windows Forms.NET 982

Windows Service Applications 801

WindowsIdentity 741

WinMain 376

X

x509 729

X509 700

X509Certificate2 730

X509Store 730

XCopy Deployment 318

Xml Schema Definition 904

XmlException 254

XmlIgnore 549

XmlRoot 402

XmlSerializer 540

Y

YACC 69

Yazılım 9

yield 435

1. VB.NET için karakter dizi birleştirme işleminde her iki operatör de kullanılabilir. [↑](#footnote-ref-1)
2. Desteklenmemektedir. [↑](#footnote-ref-2)
3. Desteklenmemektedir, System.Nullable.HasValue ile benzer işlem yapılabilir. [↑](#footnote-ref-3)
4. Desteklenmemektedir. [↑](#footnote-ref-4)
5. C# için herhangi bir niteleyici kullanmaya gerek yoktur. [↑](#footnote-ref-5)
6. Sadece C# tarafından desteklenmektedir. [↑](#footnote-ref-6)
7. Desteklenmemektedir. [↑](#footnote-ref-7)
8. Sadece C# tarafından desteklenmektedir. VB.NET ile Win32 DLL dosyalarının kullanımı hakkında bir sonraki konuda gerekli örnek gösterilmiştir. [↑](#footnote-ref-8)
9. Kitabın ilerleyen bölümlerinde detaylı bilgi verilecektir. [↑](#footnote-ref-9)