T.C.
MİLLİ EĞİTİM BAKANLIĞI

MEGEP

(MESLEKİ EĞİTİM VE ÖĞRETİM SİSTEMİNİN GÜÇLENDİRİLMESİ PROJESİ)

BİLİŞİM TEKNOLOJİLERİ

NESNE TABANLI PROGRAMLAMA - 1

ANKARA 2008

Milli Eğitim Bakanlığı tarafından geliştirilen modüller;

- Talim ve Terbiye Kurulu Başkanlığının 02.06.2006 tarih ve 269 sayılı Kararı ile onaylanan, Mesleki ve Teknik Eğitim Okul ve Kurumlarında kademeli olarak yaygınlaştırılan 42 alan ve 192 dala ait çerçeve öğretim programlarında amaçlanan mesleki yeterlikleri kazandırmaya yönelik geliştirilmiş öğretim materyalleridir (Ders Notlarıdır).
- Modüller, bireylere mesleki yeterlik kazandırmak ve bireysel öğrenmeye rehberlik etmek amacıyla öğrenme materyali olarak hazırlanmış, denenmek ve geliştirilmek üzere Mesleki ve Teknik Eğitim Okul ve Kurumlarında uygulanmaya başlanmıştır.
- Modüller teknolojik gelişmelere paralel olarak, amaçlanan yeterliği kazandırmak koşulu ile eğitim öğretim sırasında geliştirilebilir ve yapılması önerilen değişiklikler Bakanlıkta ilgili birime bildirilir.
- Örgün ve yaygın eğitim kurumları, işletmeler ve kendi kendine mesleki yeterlik kazanmak isteyen bireyler modüllere internet üzerinden ulaşılabilirler.
- Basılmış modüller, eğitim kurumlarında öğrencilere ücretsiz olarak dağıtılır.
- Modüller hiçbir şekilde ticari amaçla kullanılamaz ve ücret karşılığında satılamaz.

İÇİNDEKİLER

AÇIKLAMALAR ii
GİRİŞ 1
ÖĞRENME FAALİYETİ–1 3
1. NESNE TABANLI PROGRAMLAMA 3
1.1. Nesne Tabanlı Programlama Nedir? 3
1.1.1. Kapsülleme (Encapsulation) 4
1.1.2. Kalıtım (Inheritance) 5
1.1.3. Çok Biçimlilik (Polymorphism) 7
1.2. Net Framework 7
1.2.1. Bu Platformu Nasıl Edineceğim? 8
1.2.2. NET'in Çekirdek Kurulumu (21MB) 8
1.2.3. NET Framework SDK (134 MB) 8
1.2.4. Hangi İşletim Sistemlerinde Çalışıyor? 9
1.2.5. .Net Framework'ün Yapısı 9
1.3. MSIL (Microsoft Intermediate Language) 9
1.4. Assembly Kavramı 10
1.5. JIT Derleyiciler (Just in Time) 11
1.6. The Base Class Library - Temel Sınıf Kütüphaneleri 11
1.7. Türler (TYPES) 12
1.7.1. Referans Türleri 14
UYGULAMA FAALİYETİ 16
ÖLÇME VE DEĞERLENDİRME 17
ÖĞRENME FAALİYETİ–2 18
2. SINIFLAR (Classes) 18
2.1. Sınıf, Üye ve Nesneleri 19
2.1.1. Sınıf Üyeleri 21
2.2. İç İçe Tipler 21
2.3. Kullanıcı Tanımlı Tipler 22
2.4. Metotlar ve Dönüş Değeri Tipi 22
2.5. Erişim Türleri 24
2.5.1. Erişim Belirleyicileri 24
2.5.2. Erişim Türleri 24
UYGULAMA FAALİYETİ 31
ÖLÇME VE DEĞERLENDİRME 32
MODÜL DEĞERLENDİRME 33
CEVAP ANAHTARLARI 34
SÖZLÜK 35
ÖNERİLEN KAYNAKLAR 36
KAYNAKÇA 37

AÇIKLAMALAR

KOD	**482BK0074**
ALAN	**Bilişim Teknolojileri**
DAL/MESLEK	**Veri Tabanı Programcılığı**
MODÜLÜN ADI	**Nesne Tabanlı Programlama - 1**
MODÜLÜN TANIMI	Nesneye yönelik programlama dilinin çevresini hazırlayabilme ve sınıf, metot ve yapılar ile çalışabilme ile ilgili bilgi ve becerilerin kazandırıldığını sağlamaya yönelik öğrenme materyalidir.
SÜRE	40/32
ÖN KOŞUL	Görsel Programlama dersi modüllerini bitirmiş olmak
YETERLİK	Nesne tabanlı programlama dilinin gerekliliklerini yerine getirmek
MODÜLÜN AMACI	**Genel Amaç** Nesneye yönelik programlama dilinin çevresini hazırlayabilecek ve sınıf, metot ve yapılar ile çalışabileceksiniz. **Amaçlar** ➢ Nesneye yönelik programlama dilinin çevresini hazırlayabileceksiniz. ➢ Sınıflar, metotlar ve yapılar ile çalışabileceksiniz.
EĞİTİM ÖĞRETİM ORTAMLARI VE DONANIMLARI	Bilgisayar laboratuvarı, bu programları kullanan işletmeler.
ÖLÇME VE DEĞERLENDİRME	➢ Her faaliyet sonrasında o faliyetle ilgili değerlendirme soruları ile kendi kendinizi değerlendireceksiniz. ➢ Öğretmen modül sonunda size ölçme aracı (uygulama, soru-cevap) uygulayarak modül uygulamaları ile kazandığınız bilgi ve becerileri ölçerek değerlendirecektir.

GİRİŞ

Sevgili Öğrenci,

Her geçen gün programların daha karmaşık bir hal alması, program kodunun kurumsal uygulama projelerinde on binlerce satırı bulması ve yazılım geliştirme maliyetinin çok artması gibi nedenlerden dolayı bilim adamları, programcılara yeni bir yaklaşımın kullanılabilineceğini öğrettiler. Bu yaklaşımın ismi Nesne Tabanlı Programlama (Object Oriented Programming) dır.

Nesne tabanlı programlama tekniği, diğer yaklaşımlara nazaran, yazılım geliştiren insanlara büyük avantajlar sağlamaktadır. Birincisi karmaşık yazılım projelerinin üretilmesini ve bakımını kolaylaştırıyor olmasıdır. Diğeri ise program kodunun tekrar kullanılabilmesine (code-reusability) olanak sağlamasıdır. Bu noktada program kodunun tekrar kullanılabilmesi profesyonel yazılım şirketlerinin maliyetlerini azaltmıştır. Dolayısı ile programların lisans ücretleri düşmüş ve sektörün sürekli olarak canlı kalmasına ve rekabet içinde gelişmesine yardımcı olmuştur.

Bu bölümde günümüz programlarının bel kemiğini oluşturduğuna inandığımız nesne tabanlı programlamadan bahsedilecektir.

ÖĞRENME FAALİYETİ–1

AMAÇ

Uygun ortam sağlandığında hatasız bir şekilde programlama dilinin çevresini hazırlayacaksınız.

ARAŞTIRMA

Bu faaliyet öncesinde aşağıda belirtilen araştırma faaliyetlerini yapmalısınız:

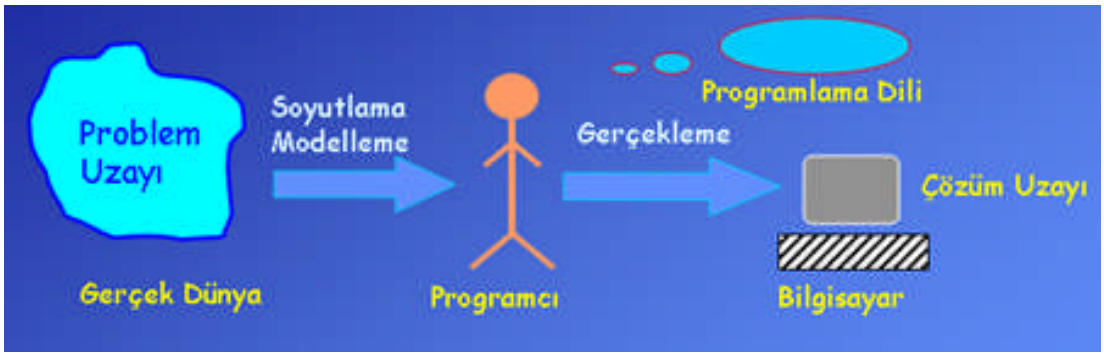
- İlk zamanlarda kullanılan programlarda nesne diye bir kavram var mıydı?
- Nesne tabanlı programlamaya neden ihtiyaç duyulmuştur?
- Günümüzde kullanılan programlama dillerinden hangisi nesne tabanlı programlamayı desteklemektedir?

1. NESNE TABANLI PROGRAMLAMA

1.1. Nesne Tabanlı Programlama Nedir?

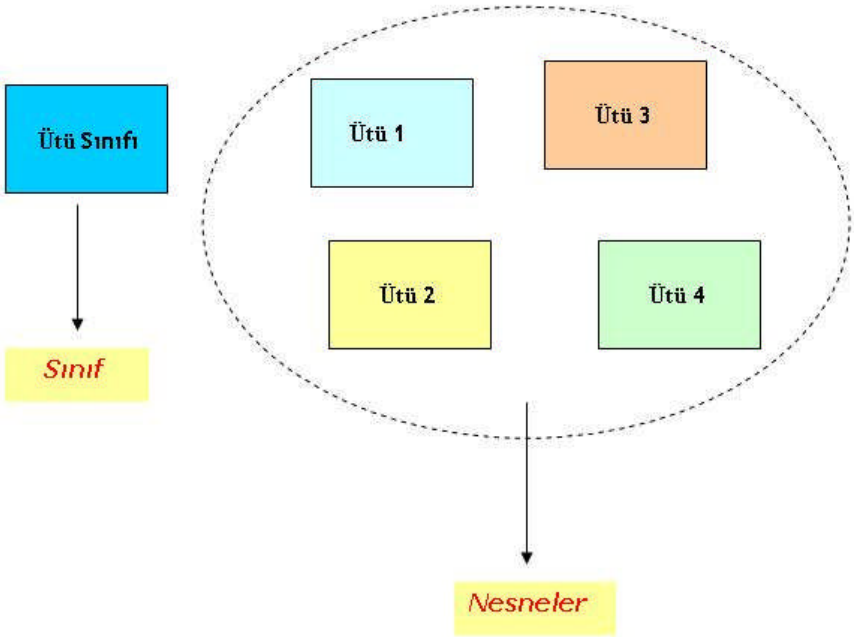
Nesne tabanlı programlamada esas olan, gerçek hayatta var olan olguların programlamaya aktarılmasındaki yeni yaklaşımdır. Klasik prosedürel programlamada verilerimiz ve fonksiyonlarımız vardı. Yani her şey veri ve bu veriyi işleyen metotlar etrafında dönüyordu.

Aslında nesne tabanlı programlamada da iki önemli birim; veri ve veriyi işleyip mantıklı sonuçlar üreten metotlar bulunur. Ama buradaki fark gerçek hayattaki olguların da daha iyi gözlenip programlama dünyasına aktarılmasındadır.



Resim 1.1: Nesne tabanlı programlama modeli

Mesela elimizde bir ütümüz olsun. Ütünün markası, modeli, rengi, çalıştığı elektrik voltajı, ne tür kumaşları ütüleyebildiği bu ütüye ait özelliklerdir (veri). Aynı zamanda ütümüzü ısıtabiliriz, ütüleme işinde kullanabiliriz ve soğumaya bırakabiliriz. Bunlar ise ütünün fonksiyonlarıdır (metot).



Resim 1.2: Sınıf (Class) ve nesne (Object) kavramı

Eğer ütü ile ilgili bir program yapmış olsak ve nesne tabanlı programlama tekniğini kullansak hemen bir ütü sınıfı (class) oluştururduk. Bu sınıfta ütüye ait bilgiler (veriler) ve ütü ile yapabileceğimiz işler (metot) bulunurdu. O zaman nesne tabanlı programlamada bir sınıfta, sınıfa ait veriler ve bu verileri işleyip bir takım faydalı sonuçlar üreten fonksiyonlar / metotlar bulunur.

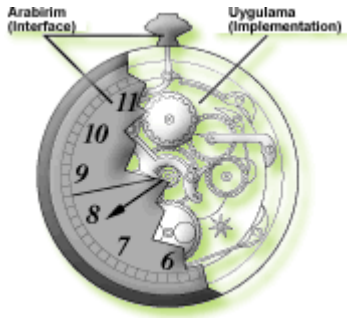
Dahası, biz bir tane ütü sınıfı tasarlarsak bu sınıftan istediğimiz sayıda değişik ütüler (Object veya instance) yapabiliriz. Resim 1.2'de bu şekilde ütü sınıfı ve bu sınıftan oluşturduğumuz nesnelerin görsel olarak anlatımı bulunmaktadır.

Nesne tabanlı programlamanın bazı temel kavramları vardır. Şimdi bu kavramlardan bahsedelim.

1.1.1. Kapsülleme (Encapsulation)

İşlemlerin ve verilerin gizlenmesine veya bir grup içerisinde toplanmasına kapsülleme denir. Kapsüllleme sayesinde, sınıfımız içinde kullandığımız alanların dışarıdan herhangi bir etki ile doğrudan değiştirilmelerini engellemiş oluyoruz. Bu durum bizim isteğimiz dışında oluşabilecek atamaların önüne geçmemize ve ille de bu alanların değerleri değişecekse bizim belirlediğimiz bir çizgide olmasına neden oluyor. Bu da programcılıkta ve özellikle nesne

tabanlı programlamada çok büyük bir yere sahip olan kontrolün elimizde olmasını sağlıyor.

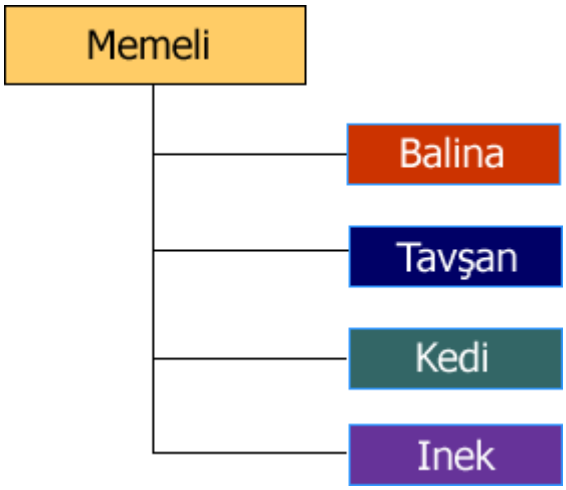


Yandaki resimde gördüğünüz saat içinde çok karmaşık mekanizmalar bulunmaktadır. Ama bu karmaşık mekanizma saatin kullanıcısını ilgilendiren bir olay değildir. Kullanıcı için önemli olan saatin düzgün çalışmasıdır.

Resim 1.3: Kapsülleme

1.1.2. Kalıtım (Inheritance)

Normalde bu kavramı gerçek hayattan biliyoruz. En basit anlamda, örneğin “Ben, annemin gözlerini almışım.” dediğimde, tıp uzmanlarının buna getirdikleri yorum " siz annenizden kalıtımsal olarak şu özelikleri almışsınız." oluyor. Programlama dillerinde de kalıtımın rolünün aynı olduğunu söyleyebiliriz. Zaten nesne tabanlı programlama dillerini tasarlayan uzmanlar, gerçek hayat problemlerini, bilgisayar ortamına taşıyabilmek amacıyla en etkili modelleri geliştirmişler, bu model içerisine kalıtımı da katarak çok önemli bir özelliğin kullanılabilmesini sağlamışlardır.

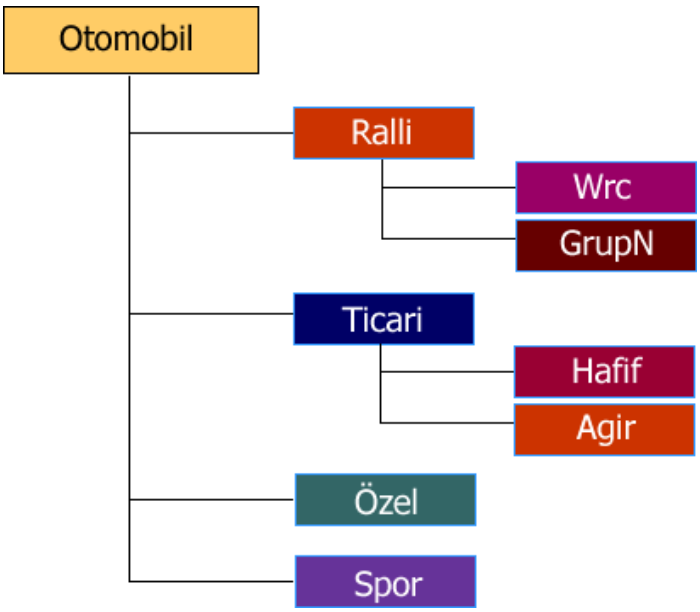


Resim 1.4: Kalıtım (Inheritance) kavramı

Kalıtım kavramının programlama dilleri içerisinde bir tanımını yapmak lazım. En genel tanımı ile kalıtım, **"Bir sınıftan yeni sınıflar türetmektir"** Herşeyden önce kalıtım yolu ile bir sınıftan yeni sınıflar türetmenin, türetilen sınıflara etkisi nedir? Bu sorunun cevabı kalıtımın da özünü oluşturmaktadır. **Türetilen her bir sınıf, türediği sınıfın özelliklerini de devralır**. Buradan, türetilmiş bir sınıf içerisinden, türediği sınıfa ait üyelere

erişilebileceği sonucunu çıkartabiliriz. Elbette bu erişimin de bazı kuralları vardır. Örneğin erişim belirleyicilerinin etkisi veya aynı üyelerin kullanılışı gibi durumlar.

Neden bir sınıftan başka sınıflar türetiriz? Bunun cevabı şudur: Tüm sınıflarda ortak olan özellikleri tek bir sınıf içerisinde toparlamak. Bu modellerimizi geliştirirken, her sınıf için ortak olan üyelerin tekrar yazılmasını engellemekle kalmayacak, sınıflar arasında düzenli bir hiyerarşi yapısının oluşmasını da sağlayacaktır.



Resim 1.5: Kalıtım kavramına bir başka örnek

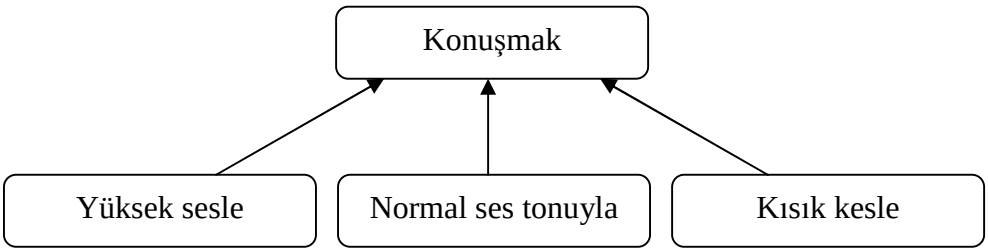
Şimdi güzel bir örnek verelim. Gerçek hayat modelleri bu iş için biçilmiş kaftantır. Örneğin, otomobilleri bir temel sınıf olarak düşünebiliriz.

Bu sınıftan otomobillere ait değişik kategorileri türetebiliriz.(Resim 1.5'e bakınız) Buradaki tüm sınıfların ortak bir takım özellikleri var. Bir motorlarının olması, tekerleklerinin olması, viteslerinin olması vb. ama aynı zamanda her ayrı sınıfın kendine has özellikleri de vardır. Örneğin ralli araçları için güvenlik bariyerlerinin olması, pilotlar için kaskların kullanılması gibi. Bu tabloyu inceleyince "Her ralli aracı bir otomobildir" diyebiliriz. Bu ralli araçlarının otomobil sınıfından türediğini gösterir. Diğer yandan "Her WRC bir ralli aracıdır." da diyebiliriz. Bu ise, WRC araçlarının ralli araçlarının birtakım ortak özelliklerine sahip olduğunu ayrıca otomobillerin de bir takım ortak özelliklerine sahip olduğunu gösterir.

İlk aşamada ralli, ticari, özel ve spor sınıflarının otomobil sınıfından türediğini söyleyebiliriz. Bununla birlikte WRC ve GrupN sınıflarıda otomobil sınıfından türeyen ralli sınıfından türemiştir. Yani burada şunu söyleyebilmek mümkündür. WRC sınıfı hem ralli sınıfının hem de otomobil sınıfının özelliklerine kalıtımsal olarak sahiptir.

1.1.3. Çok Biçimlilik (Polymorphism)

Bir metodun farklı nesnelerde veya farklı ortamlarda farklı sonuçlar üretmesidir. Mesela “konuşmak” insan sınıfına ait bir metottur.

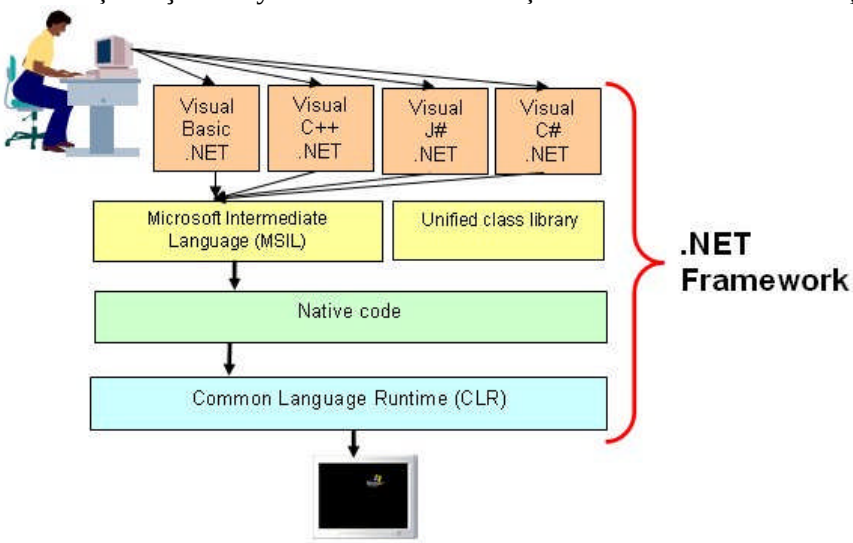


Resim 1.6: Çok biçimlilik (Polymorphism) şeması

Ancak bu metot ait olduğu insan sınıfının örneğine göre farklı sonuçlar üretir. Arkadaşımızla konuşurken farklı bir tonda, patronumuzla konuşurken farklı bir tonda konuşuruz, diğer bir ifadeyle bu metodun birden fazla biçimi vardır.

1.2. Net Framework

NET Framework, Microsoft tarafından geliştirilen, açık İnternet protokolleri ve standartları üzerine kurulmuş komple bir uygulama geliştirme platformudur. Buradaki uygulama kavramının kapsamı çok geniştir. Bir masaüstü uygulamasından tutun, bir web tarayıcı uygulamasına kadar her şey bu platform içinde düşünülmüştür ve desteklenmiştir. Bu uygulamaların birbirleriyle ve geliştirildiği ortam fark etmeksizin dünyadaki tüm uygulamalarla iletişimi için kolayca Web Servisleri oluşturulmasına imkân verilmiştir.



Resim 1.6: .NET Framework’ün bileşenleri

Bu platform, işletim sisteminden ve donanımdan daha üst seviyede taşınabilir olarak tasarlanmıştır. Özet olarak bakıldığında önümüzde duran .NET Framework :

- İnterneti hedef alan bileşen setiyle,
- Programcıları özgürleştiren, diller arası etkileşime olanak tanıyan bağımsız dil mimarisiyle,
- Donanım-bağımsız ara seviye derleme ve güvenli talimat icraatını güvence altına alan çalışma zamanı ortamıyla,
- Web uygulamalarına getirdiği özgün ve radikal yaklaşımıyla,
- Zengin ve mükemmel organize edilmiş sınıf kütüphanesi ve dokümantasyonu ile
- Sınırları ortadan kaldıran web servisi desteğiyle,

ve daha da sayabileceğimiz birçok özelliğiyle benzetildiği teknolojileri geride bırakan inkâr edilemeyecek seviyede özgün bir platformdur.

1.2.1. Bu Platformu Nasıl Edineceğim?

.NET platformu için uygulama geliştirmek istiyorsanız, işletim sisteminize. NET çalıştırma ortamını kurmanız gerekiyor. Bu ortamı, Java uygulamalarını çalıştırmak için gerekli olan Java Runtime Environment (JRE) gibi düşünebilirsiniz.

.NET platformu iki ayrı kurulum dosyası halinde, ücretsiz sunuluyor:

1.2.2. NET'in Çekirdek Kurulumu (21MB)

Bu kurulum ile bilgisayarınızda bir metin editörü açıp kodunuzu yazabilir, gelen derleyiciler ile derleyip uygulamanızı sınayabilirsiniz. Ancak bu kurulum, geliştiriciler için değil, daha çok dağıtım sonrası uygulamanızın çalışacağı sistemler için uygundur. .NET uygulamaları için gerekli çalışma zamanı ortamını, web uygulamalarının sunulabilmesi için gerekli olan Web sunucu (IIS) eklentilerini ve desteklenen temel dillerin derleyicilerini sunmaktadır. Bir yazılım geliştiricinin bunlardan daha fazlasına ihtiyacı vardır. Hele de uygulama geliştireceği ortamda herhangi bir deneyimi yoksa. Bu nedenle ikinci bir seçenek sunuluyor.

1.2.3. NET Framework SDK (134 MB)

SDK yani Software Development Kit, çekirdek kurulumu içeriğinden daha kapsamlı bir kurulumdur. Peki, 134 MB’lik kurulumda fazladan neler var: Faydalı araçlar, hata ayıklayıcılar, SQL Server’ın hafif versiyonu MSDE, hızlıca başlamanızı temin edecek QuickStart Tutorials, örnek uygulamalar ve devasa dokümantasyon. organizasyonu ve disiplini ile göz kamaştıran bir dokümantasyon olduğunu kullanmaya başladıkça çok iyi anlayacaksınız. Bu iki kurulumu da ücretsiz olarak Microsoft'un sitesinden indirebilirsiniz.

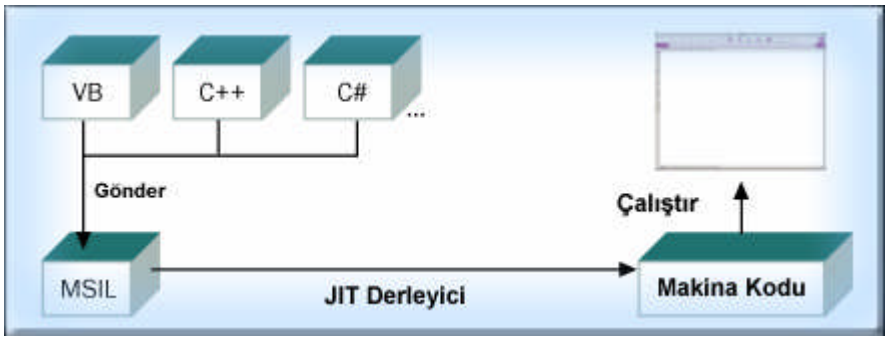
1.2.4. Hangi İşletim Sistemlerinde Çalışıyor?

Şu anda. NET Framework istemci uygulamaları Windows 98/Me/NT4/2000/XP/Server 2003 işletim sistemlerinde çalışabiliyor. Ancak sunucu davranışı için Windows 2000 (SP2 ve üstü), Windows XP veya Windows Server 2003 gerekiyor. Windows Server 2003'te Net Framework kurulmuş oluyor zaten, ayrıca yüklemenize gerek yok. Bu işletim sistemleri dışında ASP.NET sayfalarını (yani. NET'in tarayıcıları ve mobil cihazları hedef alan web uygulamaları) sunamıyorsunuz. Elbette ki bu sunum, Internet Information Server (IIS) da gerektiriyor. Teoride. NET Framework herhangi bir işletim sistemine taşınabilir. Ancak taşınacağı işletim sistemine özgü çalışma zamanı ortamının ve sınıf kütüphanesinin yazılması gerekmektedir.

1.2.5. .Net Framework'ün Yapısı

Java'dan önce, geliştirilen yazılımlar direkt olarak makine koduna çevrilirdi. Java ile program kodu önce byte code'a çevrilir. JVM (Java virtual machine) bu kodu işletim sisteminin istediği koda çevirir.

.Net içinde çalışma mantığı benzerdir .NET kodu ilk önce IL' ye (Intermediate Language-Ara dil-veya MSIL) derler, bu IL kodu çalıştırılmak istendiğinde CLR, JIT derleyicilerini kullanarak kodu makine diline çevirir.

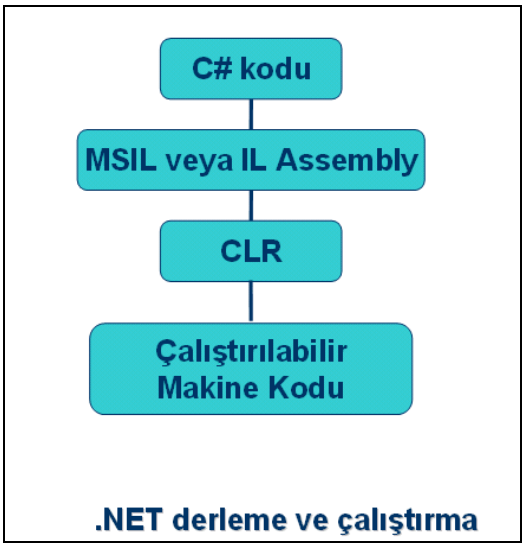


Resim 1.7: .NET Framework'ün bileşenlerini gösteren bir başka resim

1.3. MSIL (Microsoft Intermediate Language)

Normalde bir program derlendiğinde doğrudan makine koduna çevrilirken, .NET uyumlu bir dil ile derleme yapıldığında program kodu makine koduna değil de MSIL (Microsoft Intermediate Language)'e çevrilir.

MSIL işlemciden bağımsız komut setinden oluşmaktadır. Bu komut seti içerisinde nesnelerin yüklenmesi, depolanması ve initialize (başlatılması) edilmesini sağlayan komutların yanı sıra aynı zamanda nesneler üzerinde metot çağrımını sağlayan komutlar da yer almaktadır. .NET Framework ile yazılmış bir kaynak kodun derlenmesi sonucu MSIL kodu oluşmaktadır. MSIL, NET dilleri arasında bir bütünleşme sağlamaktadır. MSIL kodun çalıştırılmasından önce var olan MSIL kodu, doğal makine koduna (native code) dönüştürülmektedir. Burada interpretation dediğimiz yorumlama işlemi yapılmaz.



Resim 1.8: .Net tabanlı bir programın çalıştırılma evreleri

CLR (Common Language Runtime): Ortak Dil Çalışma Platformu olarak da adlandırılan CLR, makine diline çevrilmiş bu kodu önbellekte tutar, bu performans artışına sebep olurken diğer taraftan sistem hafızasında küçümsenmeyecek yer işgal eder.

- CLR, .NET altyapısında programların çalışmasını kontrol eden ve işletim sistemi ile programımız arasında yer alan arabirimdir. Normalde yazılan kodlar makine diline çevrilir ve işletim sistemi ile direkt bağlantı kurup çalışırdı.
- Eğer platformdan bağımsız bir ortam istiyorsak, ihtiyaç duyulan şey CLR dir, hangi platformda iseniz (Linux, Mac, Windows) CLR bu noktada devreye girer ve. NET programlarını farklı platformlarda işletim sistemine göre çalıştırır.

1.4. Assembly Kavramı

Assembly;pek çok programcı tarafından yanlış anlaşılmış bir kavramdır.Her şeyden önce, .NET 'teki Assembly kavramı ile herhangi bir mikroişlemciye ait sembolik makine dili birbirlerinden çok farklı şeyleri ifade eder.

.Net'te derlenen sınıf kütüphaneleri (class library 'ler) ismine Assembly denilendll dosyalarını oluşturur. Bu anlamda bir Assembly, bir ya da daha fazla fiziksel dosyayı barındıran bir koleksiyondur. Bu dosyalar; assembly içerisindeki derlenmiş sınıfların kodları, resimler, resource dosyaları ve diğer binary dosyalardır.

Assembly dosyaları .dll ya da .exe uzantılı olabilir. Assembly'ler kabaca COM bileşenlerine (dll'lerine)benzetilebilir. Ancak bildiğimiz COM dll 'lerinden oldukça farklı bir yapıya sahiptir. Bu farklılığın en önemli sebebi assembly’lerin registry'i kullanmamasıdır. Oysa bildiğiniz gibi, COM dll'leri kendilerine ait bir takım kimlik bilgilerini saklamak için registry'i yoğun olarak kullanmaktadır. Bileşenlerin registry kullanımı beraberinde bazı

dezavantajlar getirmektedir. Bir an için Visual Basic ile bileşen tabanlı uygulamalar geliştirdiğimiz günlere dönersek, kendi makinemizde yazdığımız ve derlediğimiz bir uygulamayı, dosya kopyalaması ile başka bir makineye taşıdığımızda uygulamanın çalışmadığını görürüz. Bunun nedeni; uygulamayı taşıdığımız makinedeki işletim sisteminin, uygulamamızın kullandığı bileşenler hakkında herhangi bir bilgiye sahip olmamasıdır.

1.5. JIT Derleyiciler (Just in Time)

C# ile IL' ye derlediğimiz programı çalıştırırken JIT derleyicileri devreye girer. Bu derleyiciler programın çalıştırıldığı sistemin ve işlemcinin anlayabileceği makine kodunu oluşturur.

Windows ortamı için 3 çeşit JIT mevcuttur.

- **Normal JIT:** IL kodu makine koduna çevrilirken default (varsayılan) olarak kullanılan derleyicidir. IL kodunu orijinal makine koduna çevirir ve önbellekte tutar. Örneğin ; program içindeki derlenmiş bir metot program akışı içinde tekrar çağrılırsa önbellekten çekilir.

- **Pre-JIT:** Tüm program kodunu makine koduna çevirip sonra çalıştıran JIT. fazla hafıza gerektirir. Programın daha hızlı çalışmasını sağlar.

- **Eco JIT:** Kısıtlı hafıza ve önbellekli sistemlerde. NET programlarının daha iyi çalışmalarını sağlamak için kullanılan derleyicidir.

1.6. The Base Class Library - Temel Sınıf Kütüphaneleri

.NET Framework'te programcıların işlerini kolaylaştırmak için birtakım hazır kütüphaneler vardır fakat C# dilinde hazır kütüphaneler mevcut değildir.

Bunun yerine Framework dediğimiz altyapıda Base Class Library adı verilen temel sınıf kütüphaneleri mevcuttur. Temel sınıf kütüphaneleri, uygulamalarınızı geliştirirken size lazım olabilecek birçok servisi ve nesneyi içinde barındıran bir sistemdir. Temel sınıf kütüphaneleri, namespaces (isim alanları)'ler içinde organize edilmişlerdir.

- **System isim alanı:** .NET çalışırken gerekli temel sınıfları içerir. Ayrıca diğer tüm sınıf kütüphaneleri de bunun içinde kümelenmiştir. System hiyerarşinin tepesinde bulunur.

- Örneğin tüm veri tabanı işlemleri için kullanılacak sınıf kütüphanesi "**System.Data**" dır.

- Bu sınıf kütüphanesi içindeki SQL ile işlemler için "System.Data.SqlClient" isim alanı mevcuttur.

- **System.Net** : HTTP ve ağ protokolleri için kullanılır.
- **System.Xml** : XML verileri ile çalışmak için kullanılır.
- **System.IO** : Dosyalara bilgi girişi, dosyadan bilgi okuma, I/O işlemleri için kullanılır.
- **System.Windows.Forms**: Windows tabanlı uygulamalarda kullanılan zengin grafik arabirimi kontrollerini içerir.

1.7. Türler (TYPES)

Tür (type) konusundan bahsetmeden önce şunu bilmeniz ve aklınızdan çıkarmamanızda fayda vardır. C# nesne tabanlı bir dildir ve her şey bir nesnedir. Nesnenin yapısını da sınıflar oluşturur. (Sınıfları ilerleyen bölümlerimizde inceleyeceğiz.)

C# da tüm sınıflar Object temel sınıfından türemiştir.

Türler C# da ikiye ayrılır: Değer türleri ve referans türleri. Değer türleri ile referans türleri arasındaki farkı anlamak, C# da değişkenlerin bellek yönetiminin nasıl çalıştığını anlamak demektir. Değer türleri ve referans türleri aynı zamanda farklı iki bellek bölgesini anlatmaktadır: Stack ve Heap bellek bölgesi. Dolayısıyla değer türleri Stack bellek bölgesi ile ilişkili, referans türleri de Heap bellek bölgesi ile ilişkilidir.

Şunu hiçbir zaman unutmamanız gerekir.

Değer türünden bir değişken Stack bellek bölgesinde tutulur, Referans türünden bir değişkenin kendisi Stack bölgesinde tutulurken temsil ettiği nesne Heap bellek bölgesinde tutulur.

Referans türlerinin ne anlama geldiğini öğreneceğiz ancak şu anda temel olarak değer türlerini anlamamız gerekmektedir.

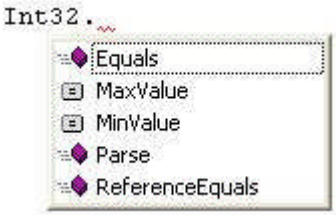
Değer türleri C# da kullanılan int, long, char gibi doğal türlerden oluşmaktadır. Net içinde tür yönetimi CTS (Common Ttype System) tarafından yönetilir. CTS. Net içinde kullanılan dillerin türlerini, dil bağımsızlığını sağlamak amacıyla ortak bir tür ile gösterir. Yani siz. Net içinde Visual Basic ya da C# ile program yazdığınızda kullandığınız türler CTS tarafından ortak bir türe dönüştürülmektedir.

Örneğin;

Visual Basic de kullanılan *integer ve* C# da kullanılan *int* türleri .Net içinde *System.Int32* ile gösterilmektedir. Peki, System.Int32 nedir? Ne anlama gelmektedir.

.Net çalışma ortamından her şey Object sınıfından türemiştir. Object sınıfından türemiş olan ValueType sınıfı, değer türleri için bir temel sınıfı oluşturmaktadır.

Değer türleri de. Net içinde bir yapı olarak (struct) tanımlanmışlardır. Dolayısıyla. Net içinde kullanılan türlerin metotları ve özellikleri vardır.



Resim 1.9: Int32 sınıfının metotları

Eğer Visual Studio içinde boş bir proje oluşturur ve Int32. yazdığımızda Int32 yapısına ait metot ve elamanları görebiliriz. Ancak şu anda biz metot, özellik, veri elemanı gibi konulara girmiyoruz. Şu anda yabancıyız. Bunlar sınıflar konusunda anlatılacaktır. Ancak aşağıdaki üç maddeyi unutmamız gerekmektedir.

- C# kullanılan doğal türlerin aslında. Net içinde CTS tarafından diğer. Net dilleri ile uyumlu olması açısından ortak bir tür ile temsil edildiği,
- C#'da değer türleri ve referans türleri olarak iki türün olduğu,
- Değer ve referans türleri ile bağlantılı stack ve heap denilen iki bellek bölgesinin olduğu unutulmamalıdır.

Aşağıdaki tabloda C#'da kullanılan türler ve bu türlerin. Net içindeki tanımları verilmiştir.

C# Türleri	**.NET içindeki tanımları**
Sbyte	System.Sbyte
Byte	System.Byte
Char	System.Char
Float	System.Single
decimal	System.Decimal
Double	System.Double
Ushort	System.UInt16
Short	System.Int16
Uint	System.UInt32
Int	System.Int32
Ulong	System.UInt64
Long	System.Int64
Bool	System.Boolean
String	System.String
Object	System.Object

Yukarıdaki tabloda gördüğümüz bool veri türü sadece *true (doğru)* ya da *false (yanlış)* değerini almaktadır.

Yani bool türünden bir değişkene sadece true veya false değeri atanmaktadır. Aksi takdirde derleyici hata verecektir.

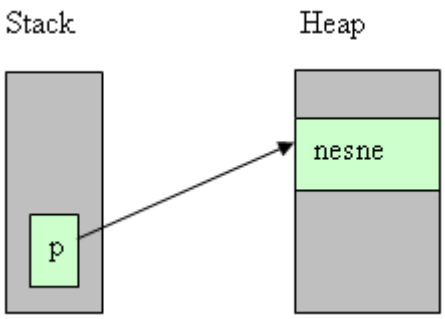
```
bool flag;
int a=1;
flag = a;
```

dediğimizde derleyici *"Cannot implicitly convert type int to bool"* şeklinde bir hata mesajı verecektir.

1.7.1. Referans Türleri

C#'da türlerin temel iki kısma ayrıldığını söylemiştik. Bunlardan birisi değer türleri (Value Types), diğeri de referans türleridir (Reference Types).

Referans türleri adres tutan türlerdir. C#'da tüm nesneler referans türündendir. Dolayısıyla C#'da her şey bir nesne olduğuna göre ve Object sınıfından türetildiğine göre demek ki C# kullanılan tüm nesneler bir adres belirtirler. Hatırlayacağınız gibi değer türleri stack bellek bölgesinde tutulurdu. Referans türünden bir değişkenin kendisi stack bölgesinde tutulurken, nesnenin kendisi heap bellek bölgesinde tutulmaktadır. Yani siz referans türünden bir değişken tanımladığınızda değişkenin kendisi stack bellek bölgesinde tutulacak ancak değer olarak nesnenin heap bellek bölgesindeki yerinin adresini tutacaktır.



Resim 1.6: Referans türlerinin hafızadaki durumu

Yukarıdaki şekildeki gibi p referans türünden bir değişkendir ve stack bellek bölgesinden tutulmaktadır. Ancak p'nin içinde nesnenin kendisi yoktur. Hatırlarsanız değer türünden bir değişken gerçek değeri tutmaktaydı. p'nin içinde nesnenin heap bellek bölgesindeki adresi vardır. Peki, niçin böyle tasarlamış olabilir? Yani niçin nesneler heap bellek bölgesinde tutulmaktadır? Bunun cevabı CLR' dir.

Çünkü bellek yönetimi CLR 'nin kontrolü altındadır. Kullanılmayan nesneler CLR yönetimindeki çöp toplayıcı mekanizma sayesinde (garbage collection) heap alanından silinmektedir.

C ve C++'da kullanılan referans gösteren & operatörü C#'da kullanılmamaktadır. Bununla birlikte göstericilerde C#'da normal olarak kullanılmamaktadır. Ancak bu C# 'ın göstericileri desteklemediği anlamına gelmez. C# göstericileri desteklemektedir eğer bir gösterici kullanmak istiyorsanız o zaman yönetilmeyen kod (unmanaged) yazmak gerekir. Tabi bu durumda karşımıza farklı bir durum çıkar.

Bellek yönetimi CLR (Common Language Runtime) yönetimindedir. Heap bellek bölgesindeki geçersiz durumdaki nesneleri temizlemektedir. Bu yönetilen koddur. Burada belleğin temizlenmesi, bellekten ayrılan yerlerin geri bırakılması tamamen CLR'nin yönetimi altındadır. O zaman yönetilmeyen kod yazdığımızda CLR aradan çıkacaktır. Bellekten ayrılan yerlerin geri bırakılması programcıya kalacaktır.

Referans türleri için heap bellek bölgesinde yer ayrımı yapıldıktan sonra geriye adres dönülmektedir. Bu adreste referans türünden bir değişkene atanmaktadır.

Bu ana kadar incelediklerimize göre:

İki tane temel tür vardır: Değer türleri ve referans türleri. Tüm doğal türler değer türüdür ve stack bellek bölgesinde tutulmaktadır. Tüm nesneler ise referans türündendir. Referans değişkenleri, nesnenin heap bellek bölgesindeki adresini tutmaktadır ve referans değişkenleri de stack bellek bölgesinde bulunmaktadır**.**

UYGULAMA FAALİYETİ

İşlem Basamakları	Öneriler
➢ Nesne tabanlı programlamayı tasvir eden bir şema çiziniz.	➢ Modülde verilen yapıyı göz önünde bulundurabilirsiniz.
➢ Kalıtım (Inheritance) ile ilgili bir örnek program parçası yazınız.	➢ Kalıtım kavramı nesne tabanlı programlamanın en önemli yapıtaşlarından biri olduğu için bu kavramı çok iyi bilmenizde fayda vardır. Modül sonundaki kaynakları kullanabilirsiniz.
➢ .NET Framework yapısını şekle dökünüz.	➢ Bir inşaatın temelinin sağlam olması ne kadar önemli ise, programlarızı yazıp derlediğimiz yapıyı bilmek de o kadar önemlidir.
➢ Kullanılan terimlerin Türkçe karşılıklarını bir kâğıda yazınız.	➢ Yabancı kelimelerin Türkçe karşılıkları modül içinde verilmiştir, ayrıca online sözlük hizmeti veren siteleri de kullanabilirsiniz.

ÖLÇME VE DEĞERLENDİRME

OBJEKTİF TEST (ÖLÇME SORULARI)

Aşağıdaki sorulardan; sonunda parantez olanlar doğru / yanlış sorularıdır. Verilen ifadeye göre parantez içine doğru ise “D”, yanlış ise “Y” yazınız. Şıklı sorularda uygun şıkkı işaretleyiniz.

1. Nesne tabanlı programada esas olan, gerçek hayattaki olguların da programlama dünyasına aktarılabilmesidir. ( )

2. IL ne demektir?
A) Interchangeable Library
B) Interoperable Languages
C) Intermediate Language
D) Interchangeable languages

3. C# derleyicisi hangi formatta çıktı verir?
A) Byte Code
B) IL
C) Hex Dump
D) Intel Assembler

4. Nesne tabanlı programlamada gizleme ve açığa çıkarma kavramını hangisi ifade eder?
A) Polymorphism
B) Encapsulation
C) Overloading
D) Static

5. Aşağıdakilerden hangisi IL (Intermediate Language) kodunu platform tabanlı koda dönüştürür?
A) MSIL-converter
B) JIT
C) JTI
D) Metadata

6. Aşağıdaki ortamlardan hangisi program kodunuzu çalıştırır?
A) MSIL
B) CLS
C) CLR
D) VB .NET

DEĞERLENDİRME

Cevaplarınızı cevap anahtarı ile karşılaştırınız. Doğru cevap sayınızı belirleyerek kendinizi değerlendiriniz. Yanlış cevap verdiğiniz ya da cevap verirken tereddüt yaşadığınız sorularla ilgili konuları öğrenme faaliyete geri dönerek tekrar inceleyiniz.

ÖĞRENME FAALİYETİ–2

AMAÇ

Uygun ortam sağlandığında sınıflar, metotlar ve yapılar ile çalışacaksınız.

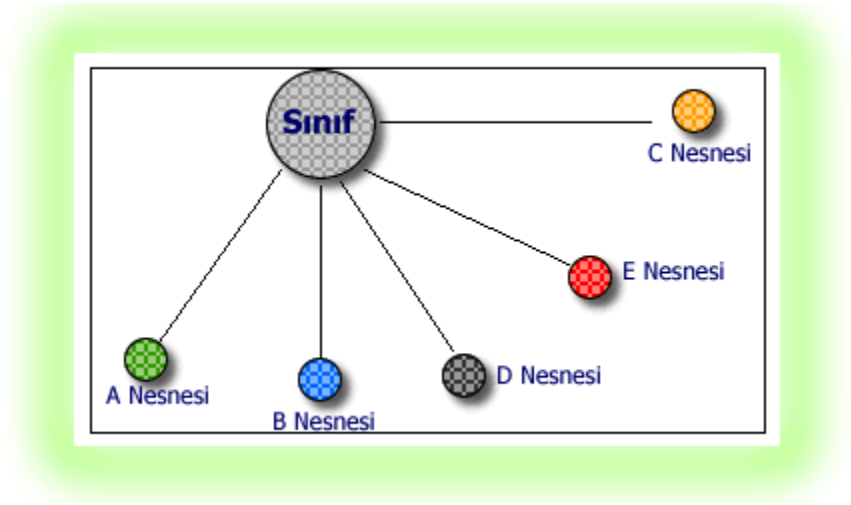
ARAŞTIRMA

Bu faaliyet öncesinde aşağıda belirtilen araştırma faaliyetlerini yapmalısınız:

- Günlük hayatta devamlı karşılaştığımız sınıflara örnekler veriniz.
- Bu sınıfların ortak ve farklı özelliklerini yazınız.

2. SINIFLAR (CLASSES)

C# dili tam anlamıyla nesne tabanlı bir dildir. Bu dil içerisinde sınıf kavramının önemli bir yeri vardır. Bu kavramı iyi anlamak, her türlü teknikte, sınıfların avantajlarından yararlanmanızı ve kendinize özgü nesnelere sahip olabilmenizi sağlar. Zaten **.NET** teknolojisinde yer alan her nesne, mutlaka sınıflardan türetilmektedir.



Resim 2.1: Sınıf (Class) ve nesne (Object) kavramı

Çevremize baktığımız zaman, çok çeşitli canlılar görürüz. Örneğin çiçekler. Dünya üzerinde kaç tür (cins) çiçek olduğunu bileniniz var mı ? Ama biz bir çiçek gördüğümüzde ona çoğunlukla "Çiçek" diye hitap ederiz özellikle adını bilmiyorsak. Sonra bu çiçeğin renginden, yapraklarının şeklinden, ait olduğu türden, adından bahsederiz. Çiçek tüm bu çiçekler için temel bir sınıf olarak kabul edilebilir. Dünya üzerindeki tüm çiçekler için ortak nitelikleri vardır. Her çiçeğin bir renginin (renklerinin) olması gibi. İşte nesne tabanlı programlama kavramında bahsedilen ve her şeyin temelini oluşturan sınıf kavramı bu benzetme ile tamamen aynıdır. Çiçek bir sınıf olarak algılanırken, sokakta gördüğümüz her

çiçek bu sınıfın ortak özelliklerine sahip birer nesne olarak nitelendirilebilir. Ancak tabiî ki çiçekler arasında da türler mevcuttur. Bu türler ise, çiçek temel sınıfından türeyen kendi belirli özellikleri dışında çiçek sınıfının özelliklerini de kalıtsal olarak alan başka sınıflardır. Bu yaklaşım Inheritance (kalıtım) kavramı olarak ele alınır ve nesne tabanlı programlamanın temel üç öğesinden biridir.

Çiçek sınıfından gerçek C# ortamına geçtiğimizde, her şeyin bir nesne olduğunu görürüz. Ancak her nesne temel olarak Object sınıfından türemektedir. Yani herşeyin üstünde bir sınıf kavramı vardır. Sınıflar, birtakım üyelere sahiptir. Bu üyeler, bu sınıftan örneklendirilen nesneler için farklı değerlere sahip olur. Yani bir sınıf varken, bu sınıftan örneklendirilmiş n sayıda nesne oluşturabiliriz. Kaldı ki, bu nesnelerin her biri tanımlandığı sınıf için ayrı ayrı özelliklere sahip olabilir.

2.1. Sınıf, Üye ve Nesneleri

Sınıf bildirimi:

Sınıf bildirimi class anahtar kelimesi ile gerçekleştirilir. Sınıf isminden sonra sınıf içindeki üye eleman ve metotlar tanımlanır.

```
class sinif_ismi
        {
        erişim veri_tipi değişken;      // Sınıf Üyeleri
        erişim dönüş_değeri
metot_ismi(parametreler)
         {
         }
        }
```

Bu örnekte class anahtar sözcüğü kullanılarak kullanıcı tanımlı (user-defined) bir sınıf (class) oluşturulmuştur. Class isminden sonra gelen sınıf üyeleri küme parantezleri ({ }) içinde yer almaktadır.

Sınıf Nesneleri Tanımlama

Örnek:

```
using System;

namespace Siniflar
{
            class Musteri
            {
                        private string Isim;

                        public void IsimAl(string isim)
                        {
                                    Isim = isim;
                        }

                        public void IsimYaz()
                        {
                                    Console.WriteLine(Isim);
                        }
            }

            class Uygulama
            {
                        static void Main ()
                        {
                                    Musteri a = new Musteri();
                                    a.IsimAl("Hakan B.");
                                    a.IsimYaz();
                        }
            }
}
```

Yukarıdaki örnekte Musteri isimli bir sınıf tanımlanmıştır. Bu sınıfın Isim adlı bir üye değişkeni, IsimAl ve IsimYaz adında iki tane yöntemi bulunmaktadır. Müsteri sınıfı, bu türden tanımlanacak bir nesnenin şemasıdır yani nesnenin verilerini ve davranışlarını göstermektedir. Daha sonra müsteri sınıfı türünden bir nesnenin bildirimi için **"new"** anahtar sözcüğü kullanılmakta ve nesnenin tanımlaması yapılmaktadır.

Bildirimler, derleyiciye değişkenler hakkında bilgi verir. Tanımlama ise bellekte bir yer ayırma işlemidir.

New anahtar sözcüğü ile sınıf için heap bellek bölgesinde bir yer ayrılmaktadır. Bu yerin adresi de yani referans değeri de stack bölgesinde saklanmaktadır. .NET Framework içinde bellek temizleme görevini üstlenmiş olan çöp toplayıcısı (garbage collector) new anahtar sözcüğü ile oluşturulmuş olan nesnelerin referansları yoksa bu alanları temizler ve belleğe geri bırakır.

2.1.1. Sınıf Üyeleri

Sınıf üyeleri temelde iki kısma ayrılmaktadır. Veri üyeleri (data members) ve fonksiyon üyeleri (member functions, methods)

- **Veri üyeleri;** alanlar (fields), sabitler (constants) ve olaylar (events)dan oluşur.

Alanlar, nesneye ait bilgilerin tutulduğu üye değişkenlerdir.

Sabitler, nesnenin değiştirilemeyen alanlarıdır.

Olaylar, belirli bir tetiklemenin meydana gelmesi sonucunda çağrılacak olan fonksiyonların adres bilgilerini tutmakla sorumludur.

Sınıfın veri üyeleri değer türlerinde olabileceği gibi, referans türlerinde de olabilir. Her nesnenin sahip olduğu data diğer nesnelerden ayrılmıştır yani her nesnenin datası kendisine aittir, hiçbir şekilde karışmaz.

- **Fonksiyon üyeleri,** yöntemler (methods), özellikler (properties), yapıcı fonksiyonlar (constructors), sonlandırıcılar (destructor), operatörler (operators) ve dizinleyiciler (indexer)den oluşur.

Yöntemler nesnenin davranış şekilleridir. Nesne kullanıcısına üye veriler üzerinde işlem yapmasına izin verir ve nesnenin dışa kapalı üyelerine ulaşılmasını sağlar.

Özellikler alanlar gibidir. Hatta bunlara akıllı alanlar da denilmektedir. Nesnenin üye verilerine değer atanmasının ve değer çekilmesinin kontrollü olarak yapılmasını sağlar.

Yapıcı ve sonlandırıcı yöntemler nesnenin özel yöntemleridir. Yapıcı yöntemler nesne ilk oluşturulduğunda çağrılırken sonlandırıcı yöntemler nesnenin ömrü tamamlandığında çağrılır.

Operatörler nesnelerin daha esnek kullanılmasına izin vermektedir.

Dizinleyiciler nesnenin bir dizi gibi kullanılmasına izin verir.

2.2. İç İçe Tipler

Tipler bazen diğer tipleri de içerebilir. Bu tür kullanımlara İç İçe Tipler adı verilir. Aşağıdaki program parçasında, bir sınıf içinde başka bir sınıf tanımlanmıştır.

```
public class KrediModulu
{
   // KrediModulu kodları buraya yazılır
   private class KrediKartlari
   {
      // KrediKartlari kodları buraya yazılır.
   }
}
```

2.3. Kullanıcı Tanımlı Tipler

Sınıflar bir veri yapısıdır. Sınıflar yardımıyla gerçek hayattaki bir nesneye yönelik modelleme yapabiliriz. Örneğin bir kredi kartı hesabı için birçok özellik vardır, hesap nu, limit, kart sahibi. Bu bizim için gerçek hayattaki bir modeli temsil eden bir yapıdır.

```
class KrediHesabi
    {
    public int HesapNo;
    public double Limit;
    public string KartSahibi;
    }
```

2.4. Metotlar ve Dönüş Değeri Tipi

Programlarımızda iş yapan en temel parçalar metotlardır. Metotlar, sınıfların üye fonksiyonlarıdır.

C, C++, Visual Basic gibi dillerde global fonksiyonlar bulunmaktaydı ancak C# da tüm fonksiyonlar bir sınıfın üyesi olmak zorundadır.

Metot bildirimi ;

```
[erişim] <dönüş değeri> metot ismi (parametre listesi)
{
        metot gövdesi;
}
```

şeklinde yapılır.

Bildirimdeki [erişim] seçeneği metoda nasıl ulaşılacağını gösterir. Eğer bir ifade yazılmazsa metot Private kabul edilir.

Yöntemler genelde public anahtar sözcüğü ile belirtilir. Böylece nesnenin kullanıcısı bu yöntemleri kullanarak nesnenin private anahtar sözcüğü ile tanımlanmış ve dışa kapalı verilerine ulaşılabilir.

C# da tüm yöntemler sınıf içinde bildirilir ve tanımlanır. C# %100 nesne tabanlı bir dil olduğundan, bir metot kullanılırken bulunduğu sınıf türünden bir nesne tanımlanır ve "." operatörü ile metoda ulaşılır.

Geri dönüş değeri olmayan metotlar “void” olarak tanımlanır. “void” tanımlı bir fonksiyon için “return” kullanılamaz.

Örnek:

```
public void BenimVoidMetodum()
{
   MessageBox.Show("Bu metot bir değer döndürmez");
}
 public int Topla(int birinci, int ikinci)
{
   int Sonuc;
   Sonuc = birinci + ikinci;
   return Sonuc;
}
```

Eğer giriş parametresi yoksa parantez içi boş bırakılır.

Örnek:

```
void yaz()
        {
        Console.WriteLine(“Elazığ”);
        return;
        }
```

“static” olarak tanımlanan metotları çağırmak için bir nesne tanımlamaya gerek yoktur. Eğer metot, içinde olduğu sınıftan çağrılacaksa metodun sadece ismi yazılarak çalıştırılabilir, eğer sınıf dışından bir çağrım yapılacaksa “SınıfAdi.Metot()” şeklinde çağrılmalıdır. Static tanımı ile oluşturulmuş metot için new operatörü ile yeni bir nesne tanımlamaya gerek yoktur.

Aşağıdaki örneği inceleyelim.

```
using System;
class Class1
{
    public static void Main()
    {
        Class1 fonk = new Class1();
        int deger = fonk.topla(3, 5);
        Console.WriteLine(deger);
        Console.Read();
    }
    public int topla(int sayi1, int sayi2)
    {
        int toplam = sayi1 + sayi2;
        return toplam;
    }
}
```

Bu örneğimizde "Class1" sınıfı için "fonk" isimli bir nesne oluşturduk ve bu nesne sayesinde "topla" fonksiyonuna ulaştık. Nesnelerin mantığını şöyle özetleyebiliriz: Nesneleri tanımlayarak sınıfların özelliklerini bir nesneye yükleriz. Ardından bu nesne ile sınıf içindeki metot ve değişkenlere (Sadece özel izin verdiklerimize) ulaşabiliriz.

Örneği inceleyecek olursak, "fonk" isimli nesnemizi tanımlamak için

Class1 fonk = new Class1() ;

kodunu yazdığımızı görürüz. Nesne tanımlamak için önce ilişki kurulacak sınıfın adı yazılır. Ardından nesnenin adı yazılır. " = " işaretinden sonra "new" komutu ile sınıf adı tekrar yazılır. Parantezlerin kullanım amacı fonksiyon ve metotlardakine benzer. Nesne tanımlandıktan sonra sınıf içindeki metot ve değişkenlere "**.**" işaretinden sonra metot veya değişken ismini yazarak ulaşabiliriz.

2.5. Erişim Türleri

2.5.1. Erişim Belirleyicileri

Nesnelerin erişim belirleyicileri için şöyle bir örnek verirsek sanırım yanlış olmayacaktır. Ben özel eşyalarımı ailemden birisi ile paylaşabilirim ancak komşularımla paylaşmam. Duruma nesne tabanlı programlama tarafından baktığımızda aile fertlerinden birisi benim özel datalarıma ulaşabilir ancak komşular ulaşamaz.

Erişim belirleyicileri ile nesne sahip olduğu üyelere kimin ve hangi şartla erişilebileceğini belirtir. Örneğin televizyonumuzun kullanıcı olarak erişebildiğimiz ve erişemediğimiz kısımları vardır (sınıfın üye elemanları). Kanal değiştirmek için kanal düğmelerini kullanırız, çünkü bunların kullanılması tamamıyla bize serbesttir. Ancak kanal bilgisinin nasıl ve nerede tutulduğunu bilemeyiz ve bu kısma erişemeyiz. Sadece kullanmaya iznimizin olduğu kısımları kullanabiliriz. Erişim belirleyicileri, nesne kullanıcısına nesneyi nasıl kullanacağını gösterir.

2.5.2. Erişim Türleri

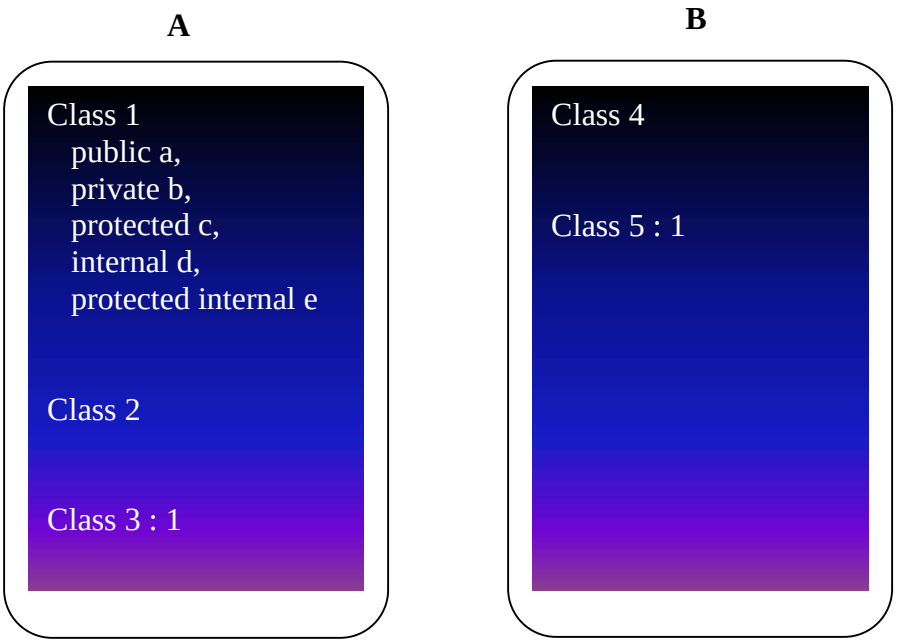
Public: Public olarak tanımlanmış bir metot veya özelliğe herhangi bir yerden ulaşmak mümkündür. Herkese açık olarak tanımlayabiliriz.

Private: Bu sıfat ile tanımlanmış metot veya özelliklere sadece tanımlandığı sınıf içerisinden ulaşabiliriz.

Protected: Protected metotlara veya özelliklere tanımlandıkları sınıftan ve bu sınıftan üretilmiş alt sınıflardan ulaşabiliriz.

Internal: Internal metot veya özelliklere tanımlandıkları sınıftan ve tanımlandıkları sınıfla aynı assemblede olan sınıflardan erişebiliriz.

Protected Internal: Bu tip metot ve özellikler hem protected hem de Internal özelliğindedir. Diğer bir ifade ile bunlara ancak kendi sınıflarından veya aynı Assembly'deki alt sınıflardan ulaşabiliriz.



Resim 2.2: Erişim türleri

Yöntemler genelde **public** anahtar sözcüğü ile belirtilir. Böylece nesnenin kullanıcısı bu yöntemleri kullanarak nesnenin **private** anahtar sözcüğü ile tanımlanmış ve dışa kapalı verilerine ulaşılabilir.

C#'da tüm yöntemler sınıf içinde bildirilir ve tanımlanır, ancak C++'da sınıfın yöntemleri, sınıfın dışında da tanımlanırdı.

Örnek:

```
using System;
namespace Siniflar
{
    class Stok
    {
        private int StokMik;

        public void SetStokMiktari(int miktar)
        {
            StokMik = miktar;
        }

        public int GetStokMiktari()
        {
            return StokMik;
        }
    }

    class Uygulama
    {

        static void Main(string[] args)
        {

            Stok st = new Stok();

            int fmiktar = 0;

            st.SetStokMiktari(10);

            fmiktar = st.GetStokMiktari();

            Console.WriteLine("Stok Miktari:{0}", fmiktar);
        }

    }

}
```

Çıktı:

```
file:///C:/Documents and Settings
Stok Miktari:10
```

Yukarıdaki örnekte stok sınıfının private olarak tanımlanmış bir veri üyesi ve public olarak tanımlanmış iki tane yöntemi vardır. Nesnenin kullanıcısı StokMik üyesine direkt olarak erişemeyeceği için public yöntemlerinden yararlanır.

Bu örnekte stok sınıfı türünden bir nesne tanımlanmış ve bu nesnenin SetStokMiktarı yöntemi ile nesnenin private veri üyesine değer atanmakta, GetStokMiktari yöntemi ile de bu değer nesne kullanıcısı tarafından alınmaktadır.

Nesnenin herhangi bir üyesine erişmek için “ **.** ” operatörü kullanılır. **st.GetStokMiktari**

Yöntemler parametreli veya parametresiz olabilir. Aynı şekilde geri dönüş değerleri de olabilir veya olmayabilir. Eğer bir yöntem geriye değer döndürmüyorsa yöntem tanımında geri dönüş değeri olmadığını belirtmek için yöntem adından önce **void** anahtar sözcüğü getirilir. Yöntem bir değer geriye döndürüyorsa yöntem tanımında nasıl bir değer geriye döndürdüğünün yöntem tanımındaki “geri_dönüş_değeri” kısmında belirtilmesi gerekir.

Metot Çağırma:

C# %100 nesne tabanlı bir dil olduğundan, bir metot kullanılırken bulunduğu sınıf türünden bir nesne tanımlanır ve “**.**” operatörü ile metoda ulaşılır. Metotlar çağrılmadıkları sürece çalışmaz.

Metotlar isimleriyle çağrılırken parametrelerden de yararlanılır.

```
// Bu satır 2 parametre ile birlikte Cevir isminde bir metot
çağırmaktadır.
Cevir(45, "Derece");
```

Eğer giriş parametresi yoksa parantez içi boş bırakılır.

“Static” olarak tanımlanan metotları çağırmak için bir nesne tanımlamaya gerek yoktur. Eğer metot, içinde olduğu sınıftan çağrılacaksa metodun sadece ismi yazılarak çalıştırılabilir, eğer sınıf dışından bir çağrım yapılacaksa “SınıfAdi.Metot()” şeklinde çağrılmalıdır.

Static tanımı ile oluşturulmuş metot için new operatörü ile yeni bir nesne tanımlamaya gerek yoktur.

Durağan ve Örnek Üyeler (Static ve Instance Member)

Bir sınıfın üyeleri durağan (static) veya örnek (instance) olabilir. Durağan olması durumunda sınıf üyesinden sadece bir tane oluşturulur, yani her nesne oluşturulduğunda durağan üyeden bir tane daha oluşturulmaz. Durağan üyeler belleğin stack bölgesini kullanır.

Tanımladığımız bir sınıfı kullanmak için o sınıftan bir nesnenin oluşturulması gerekir. Bu da nesne oluşturmadan sınıftaki herhangi bir üyeyi kullanamamız demektir. Ancak bazen öyle durumlarla karşılaşırız ki, nesne tanımlamadan da sadece sınıf adı ile kullanabileceğimiz üyelere ihtiyacımız olur. Durağan üyelere ulaşmak için nesne oluşturulmasına yani nesne örneğine gerek yoktur ve bu üyelere sınıf ismi ile erişilir. Bunun en güzel örneği main yöntemidir. "Main yöntemi" sınıfın durağan bir üyesi olduğundan, bir nesne örneğinin oluşturulmasına gerek kalmadan kullanılmakta ve programın başlangıç noktasını belirtmektedir. Kısacası durağan bir sınıf üyesi herhangi bir nesneye bağlı değildir.

Örnek üyelerin ise her nesne için bir kopyası oluşturulur ve her nesne için bu üyelerin içerikleri birbirleriyle karışmaz. Çünkü her nesne için belleğin heap bölgesinde bir yer ayrılır. Dolayısıyla her nesnenin örnek üyeleri bu bölge içinde yer alır. Örnek üyeler, durağan üyeler gibi sınıf ismi ile kullanılamaz. Bu üyelerin kullanılabilmesi için **new** anahtar sözcüğü ile bir nesnenin oluşturulması gerekir. Daha sonra nesne adı referans edilerek bu üyelere erişilir.

Durağan yöntemler sınıfın durağan olmayan üyelerine ulaşamaz. Çünkü nesnenin adresi gizli olarak nesnenin örnek üye yöntemlerine geçirilmektedir. Bu adres nesnenin bellekte bulunduğu heap bölgesinin adresidir. Durağan üyelere nesne adresi geçmediğinden nesnenin durağan olmayan üyelerine ulaşması mümkün değildir. Zaten böyle bir durumda derleme zamanında hata alınacaktır.

Nesne adresinin üye yöntemlerine geçmesi ile yöntem hangi nesneye ait olduğunu bilir. Bu adrese **this** denilir.

```
class OrnekThis
{
    private int x;
    public void Setx()
    {
        int x = 0;
        x = 10;
        this.x = 20;
    }
}
```

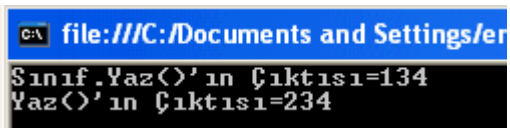
Yukarıdaki örnekte Setx yönteminin içinde local olarak tanımlanmış bir x değişkeni ve bir de this ile birlikte kullanılan x değişkeni bulunmaktadır. this.x değişkeni sınıfın private olarak tanımlanmış olan x üye değişkenini temsil etmektedir, çünkü this nesnenin adresidir.

Örnek:

```
using System;
using System.Collections.Generic;
using System.Text;

namespace fun
{
    class Sınıf
    {
        public static void Yaz(int a)
        {
            Console.WriteLine("Sınıf.Yaz()'ın Çıktısı=" + a);
        }
    }
    class Program
    {
        static void Yaz(int a) //Static tanımlama nesne için bir
bakıma global bir değer tanımı şeklinde oluyor.
        {
            a += 100;
            Console.WriteLine("Yaz()'ın Çıktısı=" + a);
            return;
        }
        static void Main(string[] args)
        {
            int a = 134;
            Sınıf.Yaz(a);
            Yaz(a);
            Console.ReadKey();
        }
    }
}
```

Çıktı:



Yapıcılar da durağan olabilir. Bu durumda sadece ilk nesne oluşturulurken çağrılacaktır. Peki, yapıcıların durağan olarak kullanılmasının sebebi ne olabilir? Bu sorunun cevabı gayet basittir: Nesnenin durağan veri üyelerine ilk değerlerini vermek.

Örnek:

```
using System;

namespace Siniflar
{
            class Base
            {
                        static Base()
                        {
                                Console.WriteLine("static yapici metod");
                        }
            }

            class Uygulama
            {
                        static void Main (string[] args)
                        {
                                    Base x1 = new Base();
                                    Base x2 = new Base();
                        }
            }
}
```

Çıktı:

```
file:///C:/Documents and Settings/
static yapici metod
```

Yukarıdaki örnekte Base sınıfından iki tane nesne tanımlanmıştır. Base sınıfının durağan bir yapıcı yöntemi bulunmaktadır. Nesnelerin new anahtar sözcüğü ile oluşturulması sırasında durağan yapıcı yöntem sadece bir kez çağrılmıştır.

Durağan yapıcı yöntemlerde diğer yapıcı yöntemlerden farklı olarak erişim belirleyicisi kullanılmaz.

Durağan yapıcı yöntemler parametresiz olarak kullanılır.

UYGULAMA FAALİYETİ

İşlem Basamakları	Öneriler
➢ Çevrenizdeki gerçek bir olgudan yeni bir sınıf yapısı oluşturunuz.	➢ Mesela araba markalarını veya cep telefonlarını düşünebilirsiniz.
➢ Tasarladığınız sınıfa ait yeni nesneler oluşturunuz.	➢ 4 kapılı araç, bluetooth özelliği olan telefon gibi.
➢ Hazırladığınız sınıfa ait iki metot tanımlayınız.	➢ Modüldeki örneklere bakınız.
➢ Sınıfa ait metotları programınızın herhangi bir yerinden çağırınız.	➢ Modüldeki örneklere bakınız.
➢ Parametre kullanarak geriye değer döndüren bir metot yazınız.	

ÖLÇME VE DEĞERLENDİRME

OBJEKTİF TEST (ÖLÇME SORULARI)

Aşağıdaki sorulardan; sonunda parantez olanlar doğru / yanlış sorularıdır. Verilen ifadeye göre parantez içine doğru ise “D”, yanlış ise “Y” yazınız. Şıklı sorularda uygun şıkkı işaretleyiniz.

1. .NET teknolojisinde yer alan tüm nesneler sınıflardan türemiştir. ( )

2. Garbage Collector (Çöp Toplayıcı)’ı en iyi hangisi ifade eder?
A)IL kodlarını Heap bölgesine taşır. B)Pointer’ları yok eder.
C) Hafıza yönetiminden sorumludur. D)Sınıf oluşturmayı sağlar.

3. Referans tipler nerede saklanır?
A) Heap bölgesinde B) Stack bölgesinde
C) Harddiskde D)Diskette

4. C# da bulunan Class lar “değer” türündendir. ( )

5. Bir sınıf özelliklerinin başka bir sınıfa aktarılması “Inheritance” olarak bilinir. ( )

6. C#’da tüm fonksiyonlar bir sınıfın üyeleri olmak zorundadır.( )

7. Static tanımı ile oluşturulmuş metot için new operatörü ile yeni bir nesne tanımlamaya gerek yoktur. ( )

8. Void tanımlı bir fonksiyon için “Return” kullanılır. ( )

9. Aşağıdaki erişim türlerinden hangisine her yerden ulaşmak mümkündür?
A) Private B) Protected
C) Public D) Internal

DEĞERLENDİRME

Cevaplarınızı cevap anahtarı ile karşılaştırınız. Doğru cevap sayınızı belirleyerek kendinizi değerlendiriniz. Yanlış cevap verdiğiniz ya da cevap verirken tereddüt yaşadığınız sorularla ilgili konuları öğrenme faaliyete geri dönerek tekrar inceleyiniz.

MODÜL DEĞERLENDİRME

PERFORMANS TESTİ (YETERLİK ÖLÇME)

Modül ile kazandığınız yeterliği aşağıdaki kriterlere göre değerlendiriniz.

DEĞERLENDİRME ÖLÇÜTLERİ	Evet	Hayır
➢ Nesne tabanlı programlamayı tasvir eden bir şema çizdiniz mi?		
➢ Kalıtım (Inheritance) ile ilgili bir örnek program parçası yazdınız mı?		
➢ .NET Framework yapısını şekle dökdünüz mü?		
➢ Kullanılan terimlerin Türkçe karşılıklarını bir kâğıda yazdınız mı?		
➢ Çevrenizdeki gerçek bir olgudan yeni bir sınıf yapısı oluşturdunuz mu?		
➢ Tasarladığınız sınıfa ait yeni nesneler oluşturunuz.		
➢ Hazırladığınız sınıfa ait iki metot tanımladınız mı?		
➢ Sınıfa ait metotları programınızın herhangi bir yerinden çağırdınız mı?		
➢ Parametre kullanarak geriye değer döndüren bir metot yazdınız mı?		

DEĞERLENDİRME

Yaptığınız değerlendirme sonucunda eksikleriniz varsa öğrenme faaliyetlerini tekrarlayınız.

Modülü tamamladınız, tebrik ederiz. Öğretmeniniz size çeşitli ölçme araçları uygulayacaktır, öğretmeninizle iletişime geçiniz.

CEVAP ANAHTARLARI

ÖĞRENME FAALİYETİ–1'İN CEVAP ANAHTARI

1	**D**
2	**C**
3	**A**
4	**B**
5	**B**
6	**C**

ÖĞRENME FAALİYETİ–2'NİN CEVAP ANAHTARI

1	**D**
2	**C**
3	**A**
4	**Y**
5	**D**
6	**D**
7	**D**
8	**Y**
9	**C**

SÖZLÜK

İsim	Okunuş	Anlam
assembly	ısem'bli	toplanti; montaj, takma, kurma
base	beys	temel, alt kisim, bir seyin üzerinde durdugu kisim
class	kläs	Sınıf, tabaka
collector	kılek'tır	Koleksiyoncu, alımcı, toplaç
common	kam'ın	müşterek, ortak; beraber yapılan
convert	kınvırt'	degistirmek, dönüstürmek; degismek
define	dîfayn'	tanımlamak, tarif etmek.
development	dîvel'ıpmınt	geliştirme; gelişme, gelişim.
encapsulation	enkepsuleyşın	kapsüllenme,sarma
framework	freymvörk	bünye, yapi
garbage	gar'bîc	Değersiz şey, çöp
heap	Hip	yığın, küme.
inheritance	înher'ıtıns	Miras, kalıt
intermediate	întırmi'diyît	Ortadaki, aradaki
implicit	împlîs'ît	ifade edilmeden anlaşılan, saklı.
initialize	inşilayz	baslangiç durumuna getirmek, baslatmak
instance	în'stıns	Örnek, kere, defa
internal	întır'nıl	İç, dahili
just in time	Cast in taym	Dinamik çeviri
language	läng'gwîc	Dil, lisan
method	meth'ıd	Yöntem, metot, usül, yol
native	ney'tîv	Yerli, doğal, doğuştan olan
nested	nestıt	İç içe
object	ab'cekt	Nesne, obje, cisim
orient	Orientıt	yönlendirmek, yöneltmek
polymorphism	polimorfizm	Çok biçimlilik
private	pray'vît	Özel, hususi, kişisel
protect	prıtekt'	Korumak, muhafaza etmek
public	Pablik	Herkese ait, açık
runtime	Rantaym	Çalışma zamanı
stack	stäk	Yığın, istif
static	stät'îk	Değişmeyen, sabit
type	Tayp	Çeşit, cins, tür
value	väl'yu	Değer, kıymet
void	Voyd	Hükümsüz, geçersiz, boş

NOT: "**ä**" karakteri "**e**" sesini vermektedir.

ÖNERİLEN KAYNAKLAR

Türkçe Kaynaklar

- http://www.msakademik.net/
- http://tr.wikipedia.org/wiki/C_Sharp_programlama_dili
- http://www.yazgelistir.com/Makaleler/1000000028.ygpx
- http://www.dotnetturk.com/whatisdotnet.aspx
- http://www.ceturk.com/forum/forum_topics.asp?FID=36&PN=0
- http://www.csharpnedir.com/

Yabancı Kaynaklar

- http://www. msdn.microsoft.com/vcsharp/
- www.csharp-station.com/Tutorial.aspx
- csharpcomputing.com/Tutorials/TOC.htm
- www.functionx.com/csharp/index.htm
- www.dotnetspider.com/
- www.publicjoe.f9.co.uk/csharp/tut.html
- www.csharpfriends.com/
- www.c-sharpcorner.com/
- www.csharphelp.com/
- www.codeproject.com/

KAYNAKÇA

- **BRADLEY L**.Jones, **Sams Teach Yourself the C# Language in 21 Days**, SAMS 2004.
- **L.THAI** Thuan, **LAM** Hoang, .**NET Framework Essentials**, 2nd Edition, O'Reilly 2002.
- **SHARP** John, **JAGGER** Jon, **Microsoft Visual CSharp.NET Step By Step** Version 2003, MS Press, kaynakçanın yılı yok
- **ZENGİN** Abdullah, **Visual C#** 2005, Nirvana Yayınları Ankara 2006.