T.C.
MİLLÎ EĞİTİM BAKANLIĞI

BİLİŞİM TEKNOLOJİLERİ

NESNE TABANLI PROGRAMLAMADA KALITIM VE ARAYÜZLER

482BK0160

Ankara, 2012

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

İÇİNDEKİLER

AÇIKLAMALAR

KOD	**482BK0160**
ALAN	**Bilişim Teknolojileri**
DAL/MESLEK	**Veritabanı Programcılığı**
MODÜLÜN ADI	**Nesne Tabanlı Programlamada Kalıtım ve Arayüzler**
MODÜLÜN TANIMI	"Nesne Tabanlı Programlama"da kalıtım ve arayüzler ile ilgili bilgi, beceri ve tutumların kazandırıldığı bir öğrenme materyalidir.
SÜRE	40/32
ÖN KOŞUL	Nesne Tabanlı Programlamada Dizi Değişkenler ve Koleksiyonlar modülünü tamamlamış olmak
YETERLİK	Kalıtımı ve arayüzleri (Interface) kullanabilmek
MODÜLÜN AMACI	**Genel Amaç** Kalıtım ve arayüzle çalışabileceksiniz. **Amaçlar** **1.** Kalıtım ile çalışabileceksiniz. **2.** Arayüz (interface) ile çalışabileceksiniz.
EĞİTİM ÖĞRETİM ORTAMLARI VE DONANIMLARI	**Ortam:** Nesne tabanlı programlama dili, bilgisayar laboratuvarı **Donanım:** Nesne tabanlı programlama yazılımını çalıştırabilecek yeterlikte bilgisayar, projeksiyon
ÖLÇME VE DEĞERLENDİRME	Modül içinde yer alan her öğrenme faaliyetinden sonra verilen ölçme araçları ile kendinizi değerlendireceksiniz. Öğretmen modül sonunda ölçme aracı (çoktan seçmeli test, doğru-yanlış testi, boşluk doldurma, eşleştirme vb.) kullanarak modül uygulamaları ile kazandığınız bilgi ve becerileri ölçerek sizi değerlendirecektir.

GİRİŞ

Sevgili Öğrenci,

Kalıtım ve arayüz, nesneye dayalı programlamada kilit kavramlardır. Herkes komutları alt alta sıralayarak program yazabilir. Ancak gerçek dünyada kullanılan sürdürülebilir, ekip çalışmasına uygun ve bakımı kolay uygulamalar geliştirmek söz konusu olduğunda programcı sayısı parmakla gösterilecek miktarda azalmaya başlar. Kaliteli, tercih edilebilir ve uzun ömürlü programlar yazabilmek için bu iki kavramı anlamak ve kullanmak son derece önem arz etmektedir. Bu noktada çok iddialı konuşarak şöyle de diyebiliriz: "Herkes şarkı söyleyebilir ama herkes yıldız olamaz!" Kalıtım ve arayüz tekniklerini uygun bir şekilde kullanabilmek yıldız programcı olmanın temel koşullarından birisidir.

Ayrıca teknoloji baş döndürücü bir hızla ilerlemektedir. Bu hıza ulaşmak için birtakım pratik yöntemlere ihtiyacınız olacaktır. Burada, yazdığınız programların modülerliğinden ve yeniden kullanılabilirliğinden söz ediyoruz. Her yeni projede her şeyi baştan yazmak yerine zamanla bir sınıf kütüphanesine sahip olmak isteyebilirsiniz.

İşte bu modül size "Ben iyi, hızlı ve kaliteli uygulamalar geliştiren bir programcı olacağım." diyen her öğrencinin mutlaka kavraması gereken kalıtım ve arayüz kavramlarını anlatmaya çalışacaktır.

ÖĞRENME FAALİYETİ–1

AMAÇ

Bu faaliyetle gerekli araç gereç ve ortam sağlandığında kalıtım ile çalışabileceksiniz.

ARAŞTIRMA

- Biyoloji dersinde yer alan "canlılar" konusunun neden canlıları tek tek değil de sınıflara ayırarak incelediğini araştırınız.
- Programlama dilinin sınıf kütüphanesini nesne tabanlı programlama yazılımı ortamındaki nesne tarayıcısı (object browser) penceresiyle (Görünüm (view) menüsünden ulaşarak) inceleyiniz. Buradaki sınıflandırmanın biyolojideki canlıların sınıflandırılmasına benzeyip benzemediğini arkadaşlarınızla tartışınız.

1. KALITIM

Kalıtım nesne tabanlı programlama dünyasında kilit öneme sahip bir kavramdır. Nesne tabanlı programlamada gerçek dünya problemlerini bilgisayar uygulamaları yardımıyla çözmek için bu problemleri modellemek gerekir. Sınıflar, modellemenin yapı taşlarıdır. İlerleyen konularda söz edileceği gibi bir okul yönetim uygulaması geliştirileceği düşünülsün. Uygulamada müdür, öğrenci ve öğretmenleri modellemek için sınıflardan (Okuldaki sınıflarla karıştırmayalım.) yararlanılır. Ancak hem müdür hem öğretmen hem de öğrenciler ortak özelliklere sahiptir. Örneğin, hepsinin de birer adı vardır. Müdür, öğretmen ve öğrenci sınıflarının hepsinde de ad şeklinde bir özellik kullanılırsa kod tekrarından nasıl kaçınılabilir? Bu mümkün olmaz. Bu noktada devreye kalıtım girmektedir. Bu sınıfların üstünde insan adında bir sınıf oluşturup müdür, öğretmen ve öğrenci sınıflarını da insandan türeterek bu problemin üstesinden gelinebilir. Hemen tüm problemleri modellerken benzer senaryolarla karşılaşılır ve kalıtımı yoğun bir şekilde kullanılır. Kod tekrarından kaçınmanın yegâne yolu kalıtımı kullanmaktır. Kalıtımı anladıktan sonra gerçek dünyada da aslında her şeyin kalıtımla ne kadar ilişkili olduğu görülür. Artık modellemeler kalıtım olmaksızın düşünemez hâle gelinir.

1.1. Temel Sınıflar ve Türetilmiş Sınıflar

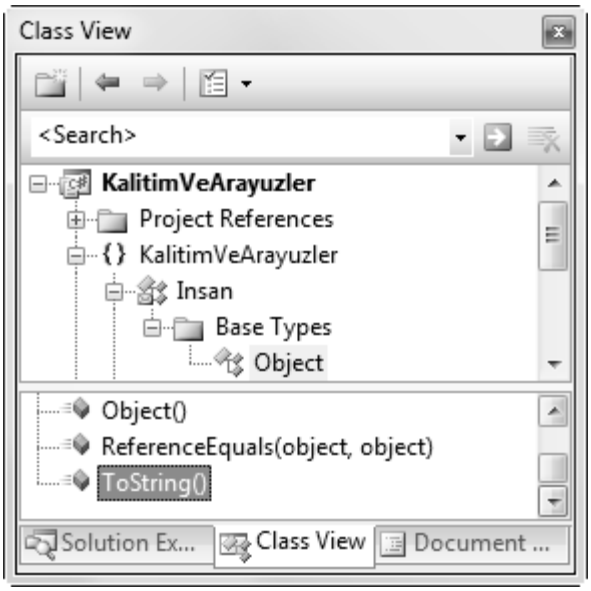
Nesne tabanlı programlama ile problemleri gerçek dünyadakine benzer tarzda nesneler şeklinde modelleyerek çözmeye çalışılır. İnşaat sektöründe bir bina nasıl inşa edilir? Önce projesi çizilir. Sonra bu proje gerçekleştirilir ve bina ortaya çıkar. Nesne tabanlı programlamada da önce sınıflar (bina örneğindeki projeye benzetilebilir) tasarlanır. Bir uygulama çok sayıda sınıf yardımıyla modellenebilir. Bu sınıflardan bazıları temel sınıf

bazıları da türetilmiş sınıflardır. Temel sınıf "Ata" sınıf olarak işlev görür. Türetilmiş sınıf ise temel sınıfın özelliklerine sahip olmanın yanında kendisi temel sınıfta bulunmayan yeni özellikler de geliştirebilir. Örneğin, girişte de bahsedildiği gibi bir "*okul yönetim uygulaması*" geliştirileceği düşünülsün. İşe okuldaki her şey modellenerek başlanır. Okulumuzda değişik rollerde insanlar vardır. Müdür, müdür yardımcısı, öğretmen, memur, hizmetli, öğrenci rolleri sayılabilir. Her insan rolü için ayrı ayrı sınıf oluşturmak yerine "*insan*" sınıfı oluşturup sonra bu sınıftan "*kalıtım*" yoluyla "*müdür*", "*müdür yardımcısı*", "*öğretmen*", "*memur*" ve "*öğrenci*" sınıfları türetilebilir. Bu örnekte "*insan*" sınıfı "*temel sınıf*", diğerleri ise "*türetilmiş sınıf*" olarak adlandırılmaktadır.

```
class Insan
{
   //temel sınıf üyeleri
}


class Ogrenci : Insan
{
   //türetilmiş sınıf üyeleri
}
```

Örnekte görüldüğü gibi "*Kalıtım yoluyla türetme* '' **:** (iki nokta üst üste) işlemci yardımıyla yapılmaktadır. İki noktanın solundaki sınıf (*Ogrenci*) türetilmiş, sağındaki ise (*Insan*) temel sınıftır. Burada *Ogrenci* sınıfı *Insan* sınıfından türetilmiş bulunmaktadır. Diğer bir ifadeyle *Ogrenci* sınıfı *Insan* sınıfının özel (private) olmayan tüm üyelerini kalıtım yoluyla almıştır. Türetme sonucunda *Insan* sınıfının özel (private) olmayan tüm alanlarını otomatik olarak *Ogrenci* sınıfı içermektedir.

Bir sınıf doğrudan en fazla bir sınıftan türetilebilir. İki ya da daha fazla sınıftan türetilemez. Eğer böyle bir ihtiyaç olursa arayüzleri kullanmak gerekir. Ancak tüm sınıflar dolaylı olarak *System.Object* sınıfından türetilmektedir. Bunun belirtilmesine de gerek yoktur. Örneğin üstte oluşturulan *Insan* sınıfına hiç üye eklenmese bile bazı üyelere otomatik olarak sahip olduğu görülür. Bunlardan bir tanesi *ToString* metodudur. Aslında bütün sınıflar bu metoda sahip olur. Bu da onların dolaylı olarak *System.Object* sınıfından türediğini kanıtlar. Aşağıdaki resimde *Class View* penceresi görülüyor. *Insan* sınıfının temel sınıfının (Base Types) Object olduğu açıkça görülmektedir.



Resim 1.1: Sınıf görünümü (class view) penceresi

Türetilmiş bir sınıftan başka sınıflar da türetilebilir. Ancak bunun bir şartı vardır. "*Sealed*" olarak tanımlanmış bir sınıftan başka bir sınıf türetilemez.

Diğer sınıflar da aynı şekilde,

```
class Mudur : Insan
{
   //türetilmiş sınıf üyeleri
}

class MudurYardimcisi : Insan
{
   //türetilmiş sınıf üyeleri
}

class Ogretmen : Insan
{
   //türetilmiş sınıf üyeleri
}

class Memur : Insan
{
   //türetilmiş sınıf üyeleri
}
```

türetilebilir.

1.2. Temel Sınıf Kurucularını Çağırma

Türetilmiş bir sınıf, temel sınıfın tüm alanlarına sahiptir. Bu alanların başlangıçta ilk değerlerinin belirlenmesi gerekir. Bu amaçla kurucu metotlar çağrılır. Türetilmiş bir sınıftan temel sınıfa ait kurucu metodu çağırmak için "*base*" anahtar sözcüğü kullanılır.

```
class Insan
{
   //temel sınıf üyeleri
   public Insan(string ad)//temel sınıf için kurucu metot
   {
      //...
   }
   //...
}

class Ogrenci : Insan
{
   //türetilmiş sınıf üyeleri
   public Ogrenci(string ad)
      : base(ad)//Insan(ad) temel sınıf kurucu metodunu çağırır
   {
      //...
   }
   //...
}
```

Bu şekilde temel sınıf kurucusu çağırılmazsa derleyici varsayılan temel sınıf kurucusunu çağırmaya çalışır. Bu da başarılamazsa bir derleyici hatası oluşur.

1.3. Sınıflar Atamak

Bir sınıf türü kullanarak bir değişken bildirmek ve sonra bir nesne yaratarak bu değişkene atamak için aşağıdaki söz dizimi kullanılır. Aşağıdaki hiyerarşik yapıya sahip olunsun.

```
class Insan
{
   //temel sınıf üyeleri
   public double Boy, Kilo;
   public string Ad;

   public Insan(string ad)//temel sınıf için kurucu metot
   {
      this.Ad = ad;
      this.Boy = 0.45;
```

```
        this.Kilo = 3.5;
    }

    public void Buyu()
    {
        Boy += 0.5;
        Kilo += 0.5;
    }
}

class Ogrenci : Insan
{
    //türetilmiş sınıf üyeleri
    public Ogrenci(string ad)
        : base(ad)//Insan(ad) temel sınıf kurucu metodunu çağırır
    {

    }

    new public void Buyu()
    {
        this.Boy += 1.5;
        this.Kilo += 1.5;
    }

    public void Oku()
    {
        //Oku metodu kodları
    }
}

class Ogretmen : Insan
{
    public Ogretmen(string ad)
        : base(ad)
    {

    }
}
```

Şöyle bir bildirim ve atama gerçekleştirebiliriz:

```
Ogrenci birOgrenci = new Ogrenci("Ali UZUN");
```

Burada *birOgrenci* değişkeni *Ogrenci* türünde bildiriliyor ve *new* sözcüğü ile oluşturulan nesne *birOgrenci* değişkenine atanıyor.

Programlama dilindeki tür denetim kuralları bir türdeki nesneyi farklı bir türde bildirilmiş bir değişkene atamaya izin vermez.

```
Ogrenci birOgrenci = new Ogrenci("Ali UZUN");
Ogretmen birOgretmen = birOgrenci; //hata
```

Yukarıdaki örnek derleme zamanı hatasına neden olur. Derleyici *Ogrenci*'nin *Ogretmen*'e açıkça dönüştürülemeyeceğini bildiren bir hata mesajı görüntüler.

Bunun yanı sıra tür olarak kalıtım hiyerarşisinin üst sırasında yer alan bir türde tanımlanmış bir değişkene alt sıradaki bir türde oluşturulmuş nesne atanabilir. Örneğin,

```
Ogrenci birOgrenci = new Ogrenci("Ali UZUN");
Insan birInsan = birOgrenci; //geçerli bir atama
```

Ogrenci sınıfından tanımlanmış *birOgrenci*, *Insan* sınıfından tanımlanmış *birInsan* değişkenine atanabilir çünkü *Insan* sınıfı hiyerarşinin üst sırasında yer almaktadır.

Aslında *Ogrenci* sınıfı *Insan* sınıfının tüm üyelerine sahip ve sadece birkaç fazlası vardır. *diplomaNotu* ve Oku üyeleridir. Yalnız üstteki örnekte *birInsan* değişkeni aracılığı ile sadece *Insan* sınıfının üyelerine erişilebilir. Yani *Ogrenci* sınıfındaki Oku metoduna erişmek mümkün olamaz.

1.4. New Metodu

Programcının çok zorlandığı durumlardan bir tanesi de sınıflara ve sınıfların üyelerine benzersiz ve anlamlı adlar bulmaktır. Eninde sonunda sınıf hiyerarşisi içerisinde temel sınıfa ait bir metodun adı türemiş sınıfta kullanılacaktır. Örneğin,

```
class Insan
{
   //temel sınıf üyeleri
   public double Boy, Kilo;
   public string Ad;

   public Insan(string ad)//temel sınıf için kurucu metot
   {
      this.Ad = ad;
      this.Boy = 0.45;
      this.Kilo = 3.5;
   }
```

```
    public void Buyu()
    {
        Boy += 0.5;
        Kilo += 0.5;
    }
}

class Ogrenci : Insan
{
    //türetilmiş sınıf üyeleri
    public Ogrenci(string ad)
        : base(ad)//Insan(ad) temel sınıf kurucu metodunu çağırır
    {

    }

    new public void Buyu()
    {
        this.Boy += 1.5;
        this.Kilo += 1.5;
    }

    public void Oku()
    {
        //Oku metodu kodları
    }
}
```

Insan temel sınıfında yer alan *Buyu* metodu *Ogrenci* türemiş sınıfında da kullanılmaktadır. Derleyici bu durumda *Insan* sınıfındaki *Buyu* metodunun gizlendiğini belirten bir uyarı görüntüler. Bu uyarı aynı zamanda "*new*" sözcüğünü kullanarak bu uyarının kaldırılabileceğini de belirtir. İşte türemiş *Ogrenci* sınıfındaki *Buyu* metodunun başında "*new*" sözcüğünü kullanma nedeni budur. Burada derleyiciye bilinçli olarak temel sınıftaki *Buyu* metodunun gizlendiği bildirilmiş olunuyor.

1.5. Virtual (Sanal) Metodu

Temel sınıfa ait bir metodun bazen türemiş sınıfta farklı bir şekilde kullanılması gerekebilir. Bu durumda temel sınıfın söz konusu metodu türemiş sınıfın değiştirip kullanabileceği yönünde izin vermesi gerekir. Bu izin, metodun temel sınıfta "*virtual*" olarak tanımlanması sonucu verilmektedir. Diğer bir ifadeyle temel sınıfta "*virtual*" olarak tanımlanmış bir metodun türemiş sınıfta yeniden yazılabileceği belirtilmiş olunuyor. Örneğin,

```
public class Sekil
{
   public const double PI = Math.PI;
   protected double x, y;
   public Sekil()
   {
   }
   public Sekil(double x, double y)
   {
      this.x = x;
      this.y = y;
   }

   public virtual double Alan()
   {
      return x * y;
   }
}
```

Sekil sınıfına ait alan adında bir metot olsun. Örnekte alan metodunun "virtual" tanımlandığına dikkat edilmelidir. Sekil sınıfından türeyecek diğer sınıflar içerisinde alan metodunun değiştirilerek yeniden yazılmasına burada izin verilmiş bulunuluyor.

1.6. Override (Geçersiz Kılma) Metodu

Temel sınıfta "*virtual*" tanımlanmış bir metodun türemiş sınıfta yeniden yazılması için (geçersiz kılma) "*override*" sözcüğü kullanılır. Temel sınıfta "*virtual*" tanımlanmamış ya da "*static*" tanımlanmış olan üyeler (metot, özellik, dizinleyici ya da olay) türemiş sınıfta yeniden yazılamaz (Geçersiz kılınamaz.). 1.4 başlığı altında verilen örneği şu şekilde genişletilebilir:

```
public class Daire : Sekil
{
   public Daire(double r)
      : base(r, 0)
   {
   }

   public override double Alan()
   {
      return PI * x * x;
   }
}
```

```
class Kure : Sekil
{
   public Kure(double r)
      : base(r, 0)
   {
   }

   public override double Alan()
   {
      return 4 * PI * x * x;
   }
}

class Silindir : Sekil
{
   public Silindir(double r, double h)
      : base(r, h)
   {
   }

   public override double Alan()
   {
      return 2 * PI * x * x + 2 * PI * x * y;
   }
}
```

Bu örnekte *Daire*, *Kure* ve *Silindir* sınıfları *Sekil* temel sınıfından türetilmişlerdir. Hepsinde de *Sekil* sınıfında "*virtual*" tanımlanan alan metodu yeniden yazılmıştır. Bu yeniden yazma (geçersiz kılma) esnasında "*override*" sözcüğünün kullanıldığına dikkat ediniz. Ayrıca burada dikkat edilmesi gereken nokta "*virtual*" ve "*override*" metotların imzalarının (dönüş türleri ile birlikte parametrelerinin tür ve sayıları) aynı olması gerekliliğidir.

Ayrıca temel sınıfta "*virtual*", "*abstract*" ya da "*override*" olarak tanımlanmış metotlar türemiş sınıfta geçersiz kılınabilir.

Bir diğer önemli nokta da "*override*" metotlar "*new*", "*static*" ya da "*virtual*" sözcükleri ile tanımlanamaz.

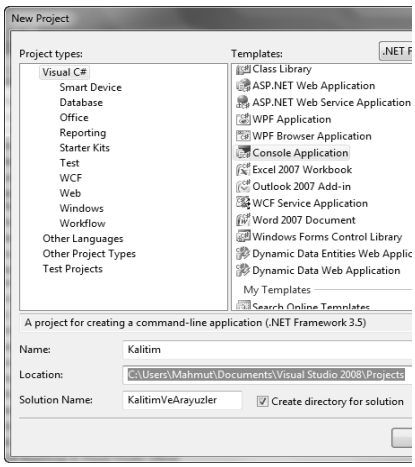
Son olarak "*virtual*" metodun erişim belirteci "*override*" sırasında değiştirilemez. Yani temel sınıfta metot "*public*" ise türemiş sınıfta da "*public*", "*protected*" ise "*protected*" kalmalıdır.

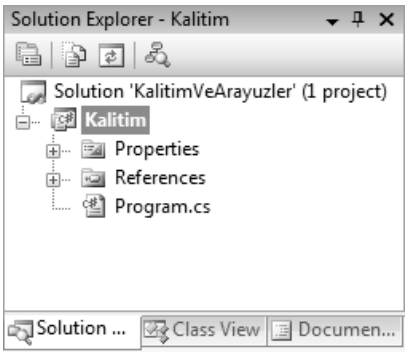
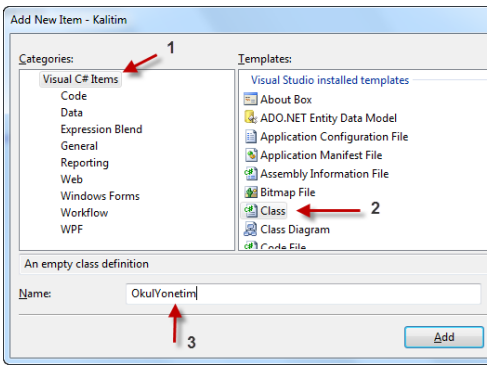
1.7. Protected (Korumalı) Erişim

"*Protected*" sözcüğü bir erişim belirtecidir. Yani bir sınıfın üyelerine kalıtım hiyerarşisi içinde o sınıftan türemiş diğer sınıfların erişebileceğini belirtir. Hiyerarşi dışından ise kimse "*protected*" üyelere erişemez. Benzer şekilde erişimi denetleyen "*public*" sözcüğü bir üyeye herkesin erişebileceğini, "*private*" sözcüğü ise sınıf dışından üyeye kimsenin erişemeyeceğini belirtir. Programlama dili tasarımcıları, sınıflar içerisinde yer alan üyelere erişim konusunda programcılara tam bir özgürlük sağlar. Ancak nesne tabanlı programlama belgeleri alanların "*private*" tutulmasını, alanlara erişimin özellik adı verilen özel metotlarla yapılmasını önerir.

UYGULAMA FAALİYETİ

Kalıtım ile çalışınız.

İşlem Basamakları	Öneriler
➢ Nesne tabanlı programlama yazılımını açınız.	➢ Başlat – Çalıştır seçeneğini tıklayınız. Çalıştır penceresi açılır. Aç kutusuna "*devenv*" yazıp Tamam düğmesine tıklayınız.
➢ Yeni bir konsol uygulaması (Console Application) projesi oluşturunuz.	➢ Dosya (File) menüsünden Yeni (New) alt menüsünü, ardından Proje… (Project…) seçeneğini tıklayınız. Yeni Proje (New Project) penceresi açılır.  **Proje types** **:** Visual C# **Templates** **:** Console Application **Name** **:** Kalitim **Location** **:** Değiştirmeyiniz **Solution Name** **:** KalitimVeArayuzler ➢ Yeni Proje (New Project) penceresinde üstteki seçim ve girişleri yaptıktan sonra Tamam düğmesini tıklayınız. Çözüm Gezgini (Solution Explorer) penceresi aşağıdaki gibi görünmelidir.

<table>
<tr>
<td></td>
<td> 
</td>
</tr>
<tr>
<td>➢ Projeye yeni bir sınıf ekleyiniz.</td>
<td>➢ Project menüsünden Add Class seçeneğini tıklayınız. Dilerseniz alternatif olarak Shift + Alt + C tuşlarına birlikte basabilirsiniz. Add New Item penceresi açılır.
 

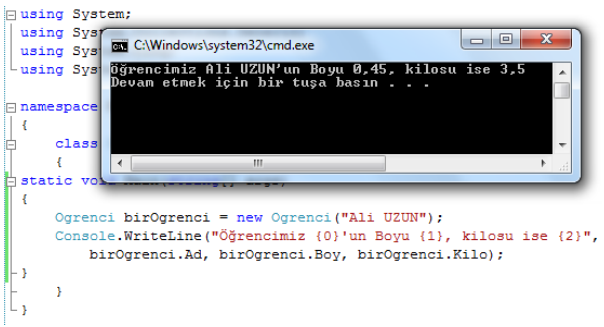
➢ Categories bölmesinden Visual C# Items'ın, Templates bölmesinden Class'ın seçili olduğundan emin olduktan sonra Name kutusuna OkulYonetim yazıp Add'i tıklayınız. Solution Explorer penceresine OkulYonetim.cs dosyası eklenip editör bölmesinde OkulYonetim.cs sekmesi açılmalıdır.</td>
</tr>
<tr>
<td>➢ OkulYonetim sınıfının adını Insan olarak değiştiriniz.</td>
<td>class OkulYonetim
{
}
sınıfını aşağıdaki şekilde olduğu gibi değiştiriniz.
class Insan
{

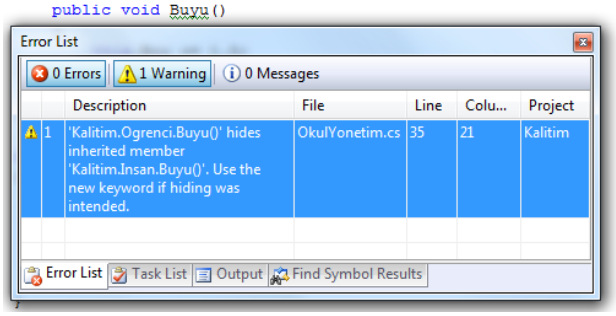
}</td>
</tr>
<tr>
<td>➢ OkulYonetim.cs dosyasına Mudur, Ogretmen ve Ogrenci sınıflarını ekleyiniz.</td>
<td>➢ Insan sınıfının küme parantezlerinin altına aşağıdaki kodları ekleyiniz.

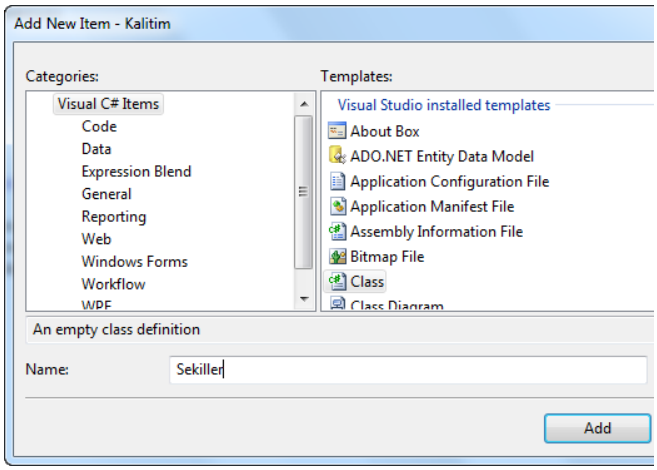
class Ogrenci : Insan
{

}</td>
</tr>
</table>

	class Ogretmen : Insan *{* *}* *class Mudur : Insan* *{* *}*
➢ *Insan* sınıfına *boy*, *kilo* ve *ad* alanları ile kurucu metot ve *Buyu* metodunu ekleyiniz.	➢ Insan sınıfını aşağıdaki şekilde olduğu gibi yeniden yazınız. *class Insan* *{* *public double Boy, Kilo;* *public string Ad;* *public Insan(string ad)* *{* *this.Ad = ad;* *this.Boy = 0.45;* *this.Kilo = 3.5;* *}* *public void Buyu()* *{* *Boy += 0.5;* *Kilo += 0.5;* *}* *}*
➢ *Ogrenci* sınıfından temel sınıf kurucusunu çağırınız.	➢ Ogrenci sınıfını aşağıdaki şekilde yeniden yazınız. *class Ogrenci : Insan* *{* *public Ogrenci(string ad)* *: base(ad)* *{* *}* *}*
➢ *Ogrenci* sınıfının bir örneğini oluşturup bir değişkene atayarak ad, boy ve kilo alanlarını ekrana	➢ Program.cs dosyası içindeki Main metodunu aşağıdaki şekilde değiştiriniz. *static void Main(string[] args)*

yazdırıp test ediniz.	*{* *Ogrenci birOgrenci = new Ogrenci("Ali UZUN");* *Console.WriteLine("Öğrencimiz {0}'un Boyu {1}, kilosu ise {2}",* *birOgrenci.Ad, birOgrenci.Boy, birOgrenci.Kilo);* *}*
➢ Uygulamayı çalıştırıp sonuçları izleyiniz.	➢ Debug menüsünden Start Without Debugging seçeneğini tıklayınız. Dilerseniz alternatif olarak Ctrl + F5 tuşlarına da basabilirsiniz. Aşağıdaki sonucu izlemelisiniz. 
➢ *Insan* sınıfındaki *Buyu* metodunu *Ogrenci* sınıfından gizleyiniz.	➢ Ogrenci sınıfını aşağıdaki şekilde yeniden yazınız. *class Ogrenci : Insan* *{* *public Ogrenci(string ad)* *: base(ad)* *{* *}* *public void Buyu()* *{* *this.Boy += 1.5;* *this.Kilo += 1.5;* *}* *}*
➢ Projeyi tekrar derleyiniz.	➢ Build menüsünden Rebuild Solution seçeneğini tıklayınız. Aşağıdakine benzer bir uyarı almalısınız.

	`` class Ogrenci : Insan { public Ogrenci(string ad) : base(ad) { } public void Buyu() ```  ➢ Bu uyarı aynı imzalı metodu türemiş sınıfta tekrar yazdığınız için Ogrenci sınıfından Insan sınıfındaki Buyu metodunu gizlediğinizi belirtir .Bu uyarıyı gizlemek için new sözcüğünü kullanabileceğiniz de bu uyarıda belirtilmektedir.
➢ *Ogrenci* sınıfındaki *Buyu* metodunun başına *new* sözcüğünü ekleyiniz.	➢ Ogrenci sınıfındaki Buyu metodunu aşağıdaki şekilde yeniden yazınız. *new public void Buyu()* *{* *this.Boy += 1.5;* *this.Kilo += 1.5;* *}*
➢ Projeyi tekrar derleyiniz.	➢ *Build* menüsünden *Rebuild Solution* seçeneğini tıklayınız. Derleyici uyarı vermekten vazgeçecektir. Burada "*new*" sözcüğü sadece uyarıyı engellemektedir.
➢ Projeye yeni bir sınıf ekleyiniz.	➢ Project menüsünden Add Class… seçeneğini tıklayın. Dilerseniz alternatif olarak Shift + Alt + C tuşlarına birlikte basabilirsiniz. Add New Item penceresi açılır.

<table>
<tr>
<td></td>
<td> 

Name kutusuna Sekiller yazıp Add düğmesine tıklayınız. Solution Explorer penceresine Sekiller.cs dosyası eklenip editör bölmesinde Sekiller.cs sekmesi açılmalıdır.
</td>
</tr>
<tr>
<td>
Sekiller.cs dosyasına Sekil, Daire, Kure ve Silindir sınıflarını ekleyiniz.
</td>
<td>
Sekiller sınıfını küme parantezleriyle birlikte sildikten sonra aşağıdaki sınıfları ekleyiniz.

<pre>public class Sekil
{
    public const double PI = Math.PI;
    protected double x, y;
    public Sekil()
    {
    }
    public Sekil(double x, double y)
    {
        this.x = x;
        this.y = y;
    }

    public virtual double Alan()
    {
        return x * y;
    }
}

public class Daire : Sekil
{
    public Daire(double r)
        : base(r, 0)
    {
    }</pre></td>
</tr>
</table>

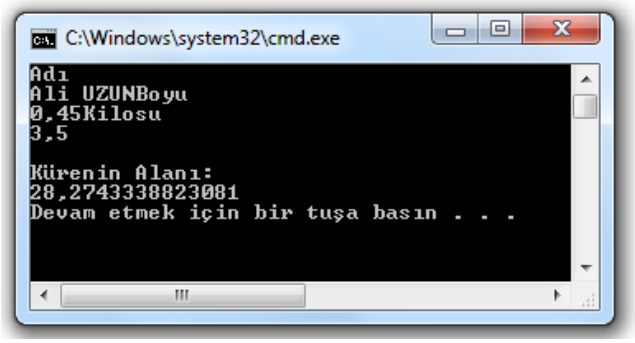
	`` public override double Alan() { return PI * x * x; } } class Kure : Sekil { public Kure(double r) : base(r, 0) { } public override double Alan() { return 4 * PI * x * x; } } class Silindir : Sekil { public Silindir(double r, double h) : base(r, h) { } public override double Alan() { return 2 * PI * x * x + 2 * PI * x * y; } } ``` ➢ *Sekil* sınıfındaki Alan metodunun "*virtual*" tanımlandığına dikkat ediniz. Aynı şekilde *Daire*, *Kure* ve *Silindir* sınıflarında da alan metotlarının "*override*" edildiğine dikkat ediniz.
➢ Main metodu içinden *Daire* sınıfının bir örneğini oluşturup alanını ekrana yazdırınız.	➢ *Program.cs* dosyasındaki main metodunu aşağıdaki şekilde değiştiriniz. ``` static void Main(string[] args) { Ogrenci birOgrenci = new Ogrenci("Ali UZUN"); Console.WriteLine("Adı"); Console.Write("{0}",birOgrenci.Ad); ```

<table>
<tr>
<td></td>
<td>

```
        Console.WriteLine("Boyu");
        Console.Write("{0}", birOgrenci.Boy);
        Console.WriteLine("Kilosu");
        Console.Write("{0}", birOgrenci.Kilo);

        Console.WriteLine("\n");
        Daire daire = new Daire(3.0);
        Console.WriteLine("Kürenin Alanı:");
        Console.WriteLine("{0}", daire.Alan());
    }
```

</td>
</tr>
<tr>
<td>➢ Uygulamayı çalıştırıp sonuçlarını izleyiniz.</td>
<td>➢ Debug menüsünden Start Without Debugging seçeneğini tıklayınız. Dilerseniz alternatif olarak Ctrl + F5 tuşlarına da basabilirsiniz. Aşağıdaki sonucu izlemelisiniz.

```
static void Main(string[] args)
{
    Ogrenci birOgrenci = new Ogrenci("Ali UZUN");
    Console.WriteLine("Adı");
    Console.Write("{0}",birOgrenci.Ad);
    Console.WriteLine("Boyu");
    Console.Write("{0}", birOgrenci.Boy);
    Console.WriteLine("Kilosu");
    Console.Write("{0}", birOgrenci.Kilo);

    Console.WriteLine("\n");
    Daire daire = new Daire(3.0);
    Console.WriteLine("Kürenin Alanı:");
    Console.WriteLine("{0}", daire.Alan());
}
```



</td>
</tr>
<tr>
<td>➢ Nense tabanlı programlama yazılımını kapatınız.</td>
<td>➢ Uygulama faaliyeti bitti. File menüsünden Exit seçeneğini tıklayarak nesne tabanlı programlama yazılımını kapatınız.</td>
</tr>
</table>

KONTROL LİSTESİ

Bu faaliyet kapsamında aşağıda listelenen davranışlardan kazandığınız beceriler için **Evet,** kazanamadıklarınız için **Hayır** kutucuklarına (X) işareti koyarak öğrendiklerinizi kontrol ediniz.

Değerlendirme Ölçütleri	Evet	Hayır
1. Temel sınıftan yeni bir sınıf türetebildiniz mi?		
2. Temel sınıf kurucusu çağırabildiniz mi?		
3. Bir sınıf tipiyle bir değişken tanımlayıp örneğini oluşturabildiniz mi?		
4. new metodu oluşturabildiniz mi?		
5. Temel sınıfta "virtual" metot oluşturabildiniz mi?		
6. Türemiş sınıfta "virtual" metodu "override" edebildiniz mi?		
7. Bir alanı "protected" olarak tanımlayabildiniz mi?		

DEĞERLENDİRME

Değerlendirme sonunda **"Hayır"** şeklindeki cevaplarınızı bir daha gözden geçiriniz. Kendinizi yeterli görmüyorsanız öğrenme faaliyetini tekrar ediniz. Bütün cevaplarınız **"Evet"** ise "Ölçme ve Değerlendirme"ye geçiniz.

ÖLÇME VE DEĞERLENDİRME

Aşağıdaki soruları dikkatlice okuyunuz ve doğru seçeneği işaretleyiniz.

1. Temel sınıf ile türetilmiş sınıf arasında kullanılan ayıraç aşağıdakilerden hangisidir?
A) : (iki nokta üst üste)
B) ; (noktalı virgül)
C) , (virgül)
D) . (nokta)

2. Bir sınıfın en fazla kaç sınıftan kalıtım yoluyla türetilebileceğinin sayısı aşağıdakilerden hangisidir?
A) 4
B) 3
C) 2
D) 1

3. Aşağıdakilerden hangisi türetilmiş bir sınıfın temel sınıftan kalıtımla alamayacağı üye için kullanılan erişim türüdür?
A) Ortak (public)
B) Özel (private)
C) İç (internal)
D) Korumalı (protected)

4. Temel sınıf kurucusunu çağırmak için kullanılan ayrılmış sözcük aşağıdakilerden hangisidir?
A) this
B) as
C) base
D) is

5. Temel sınıfta "*virtual*" tanımlanmış bir metodun türemiş sınıfta yeniden yazılması için (geçersiz kılma) hangi sözcük kullanılır?
A) override
B) new
C) base
D) void

DEĞERLENDİRME

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

ÖĞRENME FAALİYETİ–2

AMAÇ

Bu faaliyetle gerekli araç gereç ve ortam sağlandığında arayüzleri (interface) kullanabileceksiniz.

ARAŞTIRMA

- Arayüz ve kontrat kelimelerinin birbirleri ile ilişkisini araştırıp arkadaşlarınızla tartışınız.
- Bir programcı ekibi içinde yer alıyor olsaydınız yazacağınız kodlardaki standartları belirlemek için ne yapardınız? Sınıfta arkadaşlarınızla tartışınız.
- Tanıyorsanız bir programcı ekibi ile görüşerek birlikte çalışmanın zorluklarını ve nasıl çözdüklerini içeren bir söyleşi yapınız. Sonuçları bir rapor hâline getirip sınıfa sununuz.

2. ARAYÜZLER

Bir sınıftan kalıtımla almak güçlü bir mekanizmadır, asıl kalıtımın gücü bir arayüzden kalıtımla almasında yatar. Bir sınıf kalıtımla sadece bir sınıftan türetilebilir ancak bu kısıtlama kalıtımla arayüzden türetme söz konusu olduğunda ortadan kalkar yani bir sınıf birden çok arayüzden kalıtımla türeyebilir. Arayüzler sınıflar gibi kod ve veri içermez, sadece tanım içerir. Aslında soyut sınıflar arabirimlere benzerler ancak soyut sınıflar kod ve veri içerebildikleri için arayüzlerden ayrılır. Ancak çoğu zaman arayüzler soyut sınıflarla beraber kullanılarak güçlü uygulamalar geliştirilir.

2.1. Arayüz Tanımlama

Arayüz tanımlamak için class (kılas) ya da struct (sıtrakt) sözcükleri yerine “interface” (interfeys) sözcüğü kullanılır. Arayüz içinde metotları aynen bir sınıf (class) ya da yapı (struct) içerisinde olduğu gibi bildirilir. Farklı olarak herhangi bir erişim belirteci (public (pablik), private (priveyt) ya da protected (protektıd)) kullanılmaz ve metot gövdesi noktalı virgülle ayrılır. Örneğin,

```
interface IComparable
{
    int CompareTo(object obj);
}
```

Arayüz isminin başında I harfi kullanıldığına dikkat edilmelidir. Bu kullanılan sınıfın bir arayüz olduğunu anlamamıza yarayan bir isim kullanma tekniğidir. Bu sayede, sınıfların kalıtımsal olarak aldığı elemanların arayüz olup olmadığı daha kolayca anlaşılabilir.

Arayüzlerin sahip olabileceği üyeler:

- Özellikler (properties)
- Metodlar (methods)
- Olaylar (events)
- İndeksleyiciler (indexers)

Arayüzlerin sahip olamayacağı üyeler:

- Yapıcılar (constructors)
- Yokediciler (destructors)
- Alanlar (fields)

2.2. Arayüz Kısıtlamaları

Bir arayüzün hiçbir zaman bir uygulama (tanımlama dışında deyim ve ifadeler) içermeyeceği unutulmamalıdır. Bu durumun doğal sonucu olarak aşağıdaki kısıtlamalar ortaya çıkar:

- Bir arayüz için herhangi bir alan (field) tanımlanamaz (statik olsa bile). Çünkü alan bir sınıf ya da bir yapının uygulama detayıdır.
- Bir arayüz içinde kurucu (constructor) tanımlamaya da izin verilmez. Bu da uygulama detayı kabul edilir.
- Bir arayüz içinde bir yıkıcı (destructor) da tanımlanamaz.
- Bir metot için bir erişim belirteci de belirtilemez. Çünkü bir arayüzdeki tüm metotlar dolaylı olarak ortaktır (public).
- Bir arayüz içerisinde hiçbir türü yuvarlanamaz. Bir başka deyişle arayüzler sınıf, yapı ya da numaralandırma içeremez.
- Bir arayüzü bir sınıf ya da yapıdan kalıtımla alamazsınız yani türetemezsiniz. Arayüzler ,arayüzlerden türeyebilir. Yapılar ve sınıflar uygulama içerdikleri için bu kısıtlama olmasaydı üstteki bazı kısıtlamalar ihlal edilirdi.

2.3. Bir Arayüz Uygulamak

Bir arayüzü uygulamak için arayüzden kalıtımla alan ve arayüzde belirtilen tüm yöntemleri sağlayan bir sınıf ya da yapı bildirilir. Öğrenme faaliyeti -1 içinde temel sınıf olan insan hatırlanmalıdır. Bu sınıfın içerisinde Buyu adında bir metot kullanılmıştı. Insan sınıfı IInsan adında bir arayüzü uyguluyor olsun. Bu arayüz de aşağıdaki gibi tanımlanmış olsun.

```
interface IInsan
{
    void Buyu();
}
```

Arayüz adlarına I harfi ile başlamak kural olmasa bile neredeyse kural hâline gelmiş bir alışkanlıktır. Hiçbir sakıncası bulunmadığı gibi programcılar topluluğuna uyum sağlamayı kolaylaştırır.
Insan sınıfının bu arayüzü uygulaması için bu arayüzden kalıtım yoluyla türetilsin:

```
class Insan : IInsan
{
    public void Buyu()
    {
        //metot kodları
    }

}
```

Bir arayüz herhangi bir sınıfa uygulandığında (sınıf ilgili arayüzden türetildiğinde) o sınıfı tamamen kendisine benzemeye zorlayacaktır. Bu da birden fazla sınıfı aynı arayüzden türeterek bunlara tek bir değişken üzerinden erişebilme imkânı sağlayacaktır.

Örneğin, .Net, .NET Framework içerisinde yer alan Icomparable arayüzü iki nesnenin değerini karşılaştırır ve değerleri aynı ise 0, farklı ise -1 değerini üretir. Ancak bu karşılaştırmayı yapabilmesi için karşılaştırılan türlerin Icomparable arayüzünü uyguluyor olması gerekir. Bu ifade aslında bir kural tanımıdır ve şu anlama gelmektedir : "Eğer sınıftan oluşturulacak nesneleri karşılaştırılabilir yapmak isteniyorsa bu sınıfı Icomparable arayüzünden türetmelisiniz. "

Aşağıdaki örnekte IComparable arayüzünü uygulayan "int" türünden iki değer karşılaştırılmaktadır.

```
static void Main(string[] args)
{
  int s1 = 3;
  int s2 = 3;
  Console.WriteLine(s1.CompareTo(s2));
}
```

Program çalıştırıldığında ekrana 0 değerini yazacaktır. Çünkü s1 ve s2'nin değerleri aynıdır. Burada "int" türü için Icomparable arayüzünü uygular denilebilir. Siz de kendi hazırladığınız bir sınıf nesneleri için birbirleri ile karşılaştırılabilir olmasını sağlayabilirsiniz.

Aşağıdaki örnekte Daire isminde bir sınıf yazılmış ve bu sınıfa IComparable arayüzü uygulanmıştır. Böylece Daire sınıfından oluşturulan iki ayrı nesneyi karşılaştırma imkânına sahip olunur.

```
class Daire:IComparable
{
   int yaricap; //Sınıfımıza ait field(alan)

   public Daire(int yaricap)//Sınıfın kurucu metodu
   {
```

```
        this.yaricap = yaricap;
    }

    public int CompareTo(object daire1) //Arayüzden gelen metodumuz.
    {
        //metoda gelen daire1 değişkeni object olduğu için öncelikle kendi
türüne dönüştürüyoruz.
        Daire karsilastirilacak_daire = daire1 as Daire;
        //yarıçapı bu dairenin yarıçapı ile aynı ise 0, farklı ise -1
gönderiyoruz.
        return (yaricap == karsilastirilacak_daire.yaricap) ? 0 : -1;
    }
}

class Program
{
    static void Main(string[] args)
    {
        Daire daire_1 = new Daire(3);//Birinci daire nesnemiz
        Daire daire_2 = new Daire(2);//İkinci daire nesnenimiz
        //Karşılaştırılan her iki daire nesnemizinde yarıçapı faklı
olduğu için ekrana -1 yazacaktır.
        Console.WriteLine(daire_1.CompareTo(daire_2));
    }
}
```

Daire isimli sınıfımız **IComparable** arayüzünü uyguladığı için **CompareTo** isimli, dışarıdan object alan ve int değer döndüren bir metoda sahip olmalıdır. Bu kural IComparable arayüzü tarafından tanımlanmış bir kuraldır. Siz arayüzü sınıfınıza uygularsanız bu kuralı kabul etmiş sayılırsınız. Bu yüzden IComparable arayüzünden türetilmiş olunan sınıfta bu kurala uygun bir metot tanımlamak zorunda kalınır.

Bir arayüzü uygulandığında her metodun kendine uygun bir arayüz metoduyla tam olarak eşleşmesi garantiye alınmalıdır.

- Metot adları ve dönüş türleri kesin olarak eşleşmelidir.
- Parametreler (ref ve out anahtar sözcükleri dahil) tam olarak eşleşmelidir.
- Arayüz tanımı ve arayüz uygulaması arasında bir fark varsa uygulama derlenmez.
- Arayüzden türetilen sınıf içerisinde oluşturulan arayüze ait metotlar mutlaka public olarak tanımlanmalıdır(Örneğin, Daire sınıfı için CompareTo metodu private yapılıarsa derleme hatası alınır.).

Bir sınıf hem bir sınıftan devralıyor hem de bir arayüzü uyguluyorsa aşağıdaki gibi kodlanır:

```
class Ogrenci : Insan, IInsan
{
    public void Buyu()
    {
        //metot komutları
```

```
    }
}
```

Önce sınıf adı belirtilir. Bunu bir virgül takip eder. Daha sonra arayüz adı yazılır.

2.4. Bir Sınıfa Arayüz İle Ulaşma

Hiyerarşide üst sıralarda tanımlanmış bir sınıf değişkeni ile bir nesneye başvurulabildiği gibi sınıfın uyguladığı bir arayüz olarak tanımlanmış bir değişken kullanarak da nesneye ulaşılabilir. Hatırlanırsa Öğrenme Faaliyeti - 1'de

```
Ogrenci birOgrenci = new Ogrenci("Ali UZUN");
Insan birInsan = birOgrenci; //Geçerli bir atama. Ogrenci Insan'dan
türetiliyor.
```

örneğiyle Insan tipinde bildirilmiş olan birInsan değişkeni ile Ogrenci tipinde bildirilmiş olan birOgrenci nesnesine başvurmuştuk. Benzer şekilde;

```
Ogrenci birOgrenci = new Ogrenci();
IInsan birInsan = birOgrenci; //Geçerli bir atama. Ogrenci IInsan'ı
uyguluyor.
```

IInsan tipinde bildirilmiş olan birInsan arayüz değişkeni ile Ogrenci tipinde bildirilmiş olan birOgrenci nesnesine başvurulabilir. Çünkü Ogrenci sınıfı IInsan arayüzünü uygulamaktadır.
Bir nesneye bir arayüz üzerinden başvurma yöntemi faydalıdır. Çünkü türler belirtilen arayüzü uyguladığı sürece, parametre olarak farklı türler alabilen metotları tanımlayabilmeyi sağlar. Örneğin, aşağıda gösterilen Sil metodu, IInsan arayüzünü uygulayan herhangi bir bağımsız değişken alabilir.

```
static void Sil(IInsan birOgrenci)
{
    //...
}
```

Bu örnekte bağımsız değişken IInsan arayüzünü uygulayan herhangi bir nesne olabilir. Bir arayüz üzerinden bir nesneye başvurulduğunda sadece arayüz içerisinde tanımlanan metotlar kullanılabileceği unutulmamalıdır yani Ogrenci sınıfı IInsan arayüzünde tanımlanmayan Oku metoduna sahip olsaydı bu metot kullanılamayacaktı.

2.5. Soyut (Abstract) Sınıflar

IInsan arayüzünü uygulayan Ogrenci ve Ogretmen sınıflar olsun. IInsan arayüzünde de Oku metodu tanımlanmış olsun. Bu durum hem Ogretmen hem de Ogrenci iki sınıfta da tamamen aynı olacak, Oku metodunu uygulayacaklardır. Burada bir uyarı ile karşı karşıya kalmak demektir. Çünkü kod tekrar etmektedir.

```
interface IInsan
```

```
{
    void Oku();
    void Konus();
}

class Ogrenci : IInsan
{
    public void Oku()
    {
        //...
    }
    public void Konus()
    {
        //...
    }
}

class Ogretmen : IInsan
{
    public void Oku()
    {
        //..
    }
    public void Konus()
    {
        //...
    }
}
```

Oku metodunun içeriği sınıfa göre değişmeyecektir. Bir öğretmenin okuması ile öğrencinin okuması arasında ne fark olabilir Ama aynı durum Konus metodu için geçerli olmayacaktır. Çünkü Ogretmen ve Ogrenci sınıfları Konus metodunu kendilerine özgü gerçekleştireceklerdir (Herkesin ses tonu ve şivesi gibi özellikler değişkendir.).

Yinelemeden kaçınmak ve kodun bakımını kolaylaştırmak için kodu yeniden düzenlenmelidir. Bunun yolu ortak uygulamayı özellikle bu amaç için oluşturulmuş yeni bir sınıf içerisine taşımaktır. Bu tasarım şu şekilde değiştirilebilir:

```
interface IInsan
{
    void Konus();
}

class OkuyanInsan
{
    public void Oku()
    {
        //...
    }
}
class Ogrenci : OkuyanInsan, IInsan
```

```
{
}

class Ogretmen : OkuyanInsan, IInsan
{

}
```

Burada OkuyanInsan adında bir sınıf oluşturuldu ve Oku metodu buraya taşındı. Sorun çözülmüş gibi görünse de hâlâ bir problem var. OkuyanInsan sınıfının örneğini oluşturmak mümkündür. Bu çok mantıksızdır. Biz onu sadece kendisinden Ogrenci ve Ogretmeni türetmek için oluşturuldu ve ortak uygulama (Oku eylemi) sağlaması amaçlandı. Yani OkuyanInsan sınıfı kendi haklarına sahip bir girişten çok ortak işlevselliklerin bir soyutlaması olacaktır.

Bir sınıfın oluşumlarının yaratılmasına izin verilmediğini belirtmek için *abstract* anahtar sözcüğünü kullanarak açıkça sınıfın soyut olduğunu bildirmek gerekir. Örneğin,

```
abstract class OkuyanInsan
{
    public void Oku()
    {
        //Oku metodu soyut olarak tanımlanan OkuyanInsan sınıfı
içerisinde public olarak gövdesi ile birlikte oluşturuluyor.Böylece
bu sınıftan türetilen tüm sınıflar için burada yazılan "Oku" işlevi
kalıtım yolu ile geçecektir.
    }
}

class Ogrenci : OkuyanInsan, IInsan
{
  //OkuyanInsan sınıfında tanımlanan Oku metodu public olduğu için
kalıtım yolu ile Ogrenci sınıfımıza da geçecektir.

      public void Konus()//IInsan arayüzünden gelen metodum
      {
         Console.WriteLine("Soyut sınıfların nesnesi oluşturulabilir
mi?");
      }

}

class Ogretmen : OkuyanInsan, IInsan
{
 //OkuyanInsan sınıfında tanımlanan Oku metodu public olduğu için
kalıtım yolu ile Ogretmen sınıfımıza da geçecektir.

      public void Konus()//IInsan arayüzünden gelen metodum
      {
         Console.WriteLine("Soyut sınıfların nesnesi oluşturulamaz");
      }

}
```

Bir OkuyanInsan nesnesi oluşturmaya çalışılırsa kod derlenmez.

```
OkuyanInsan birInsan = new OkuyanInsan();//kod derlenmez.Abstract
olarak tanımlanan sınıflar nesnesi oluşturulmak için değil
kendisinden türetme yapılması amacıyla oluşturulmaktadır.
```

Ancak aşağıdaki kullanımlar geçerli olacaktır.

```
OkuyanInsan ogretmen = new Ogretmen();//Kalıtımdan dolayı bu kullanım
geçerlidir.
OkuyanInsan ogrenci = new Ogrenci();//Kalıtımdan dolayı bu kullanım
geçerlidir.
```

Yukarıdaki örnekte Ogretmen ve Ogrenci sınıfları hem **OkuyanInsan** soyut sınıfından hem de **IInsan** arayüzünden türetildi. Aslında soyut sınıfı biraz değiştirilerek IInsan arayüzünün işlevini de ona yaptırabilirdi.

Örnek aşağıdaki gibi değiştirilsin.

```
//IInsan arayüzünde yer alan "Konus" metodunu soyut sınıfımız içerisinde
soyut olarak işaretliyoruz.
abstract class OkuyanInsan
{
    public void Oku()
    {
        //...
    }

    public abstract void Konus();
    //abstract sınıf içerisinde abstract olarak işaretlenmiş üyeler
türeyen sınıfta mutlaka override edilmelidir. Zaten bizim isteğimiz
Ogrenci ve Ogretmen sınıfları için "Konus" metodunun olması ancak her
iki sınıf için farklı davranış göstermesidir.
}



class Ogrenci : OkuyanInsan
{
    //"OkuyanInsan" soyut sınıfımızda yer alan soyut olarak işaretlenmiş
Konus metodumuzu burada geçersiz kılıyoruz.
    public override void Konus()
    {
       Console.WriteLine("Soyut sınıfların nesnesi olabilir mi?");
    }
}

class Ogretmen : OkuyanInsan
{
    //"OkuyanInsan" soyut sınıfımızda yer alan soyut olarak işaretlenmiş
Konus metodumuzu burada geçersiz kılıyoruz.
    public override void Konus()
    {
       Console.WriteLine("Soyut sınıfların nesnesi olamaz.");
```

```
    }
}
```

Daha önceki örnekte "Konus" metodu IInsan arayüzünde tanımlanmış," Oku" metodu ise soyut sınıf olan "OkuyanInsan" sınıfı içerisinde gövdesi ile birlikte yazılmıştı.

Soyut sınıf içerisinde yazılan bir üye abstract olarak işaretlenirse arayüz içerisinde tanımlanan üye gibi davranacaktır. Yani kendisinden türetilen sınıfta bu üyenin gerçekleştirilmesini zorunlu kılar (Yukarıdaki örnekte OkuyanInsan soyut sınıfı içerisinde yer alan Konus isimli metodun Ogretmen ve Ogrenci sınıflarında gerçekleştirilmesinin zorunlu olması gibi). Bu sayede örnek için IInsan arayüzüne ihtiyaç kalmamıştır.
Soyut sınıf içerisinde yer alan bir üyenin türeyen sınıfta gerçekleştirilmesi zorunlu kılınması istenirse üyeyi abstract olarak işaretlenmelidir.
Kendi sınıf hiyerarşisi içerisinde kod tekrarını önleyecek ortak işlevselliklerin bir soyutlamasını oluşturmak için *abstract* sınıflar oluşturulabilir.

2.6. Mühürlenmiş (Sealed) Sınıflar

Kalıtımı her zaman akıllıca kullanmak çok kolay olmamaktadır. Bir arayüz ya da soyut bir sınıf oluşturursa gelecekte kalıtımla alınacak bir şeyler yazılıyor demektir. Gelecek önceden tahmin edilemez. Pratik yaptıkça ve deneyim kazandıkça esnek, kullanımı kolay arayüz, soyut sınıflar ve sınıflar hiyerarşisi oluşturma becerisi geliştirilebilir. Bu çok çaba gerektirir ve ayrıca modellediği problemi çok iyi anlamış olmak şarttır. Bunu yapmanın başka bir yolunu bulmak için temel bir sınıf olarak kullanmak üzere oluşturulmadığı sürece bir sınıftan kalıtımla alınması engellenebilir. Bunun için programlama dili *sealed* anahtar sözcüğü sunmaktadır. Örneğin,

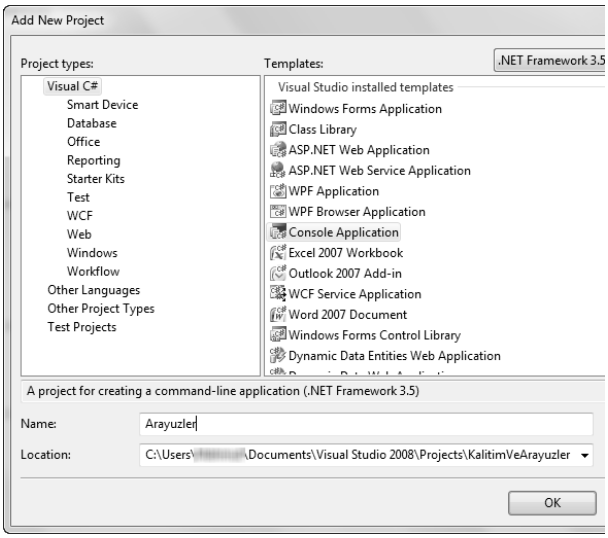
```
sealed class Ogrenci : OkuyanInsan, IInsan
{

}
```

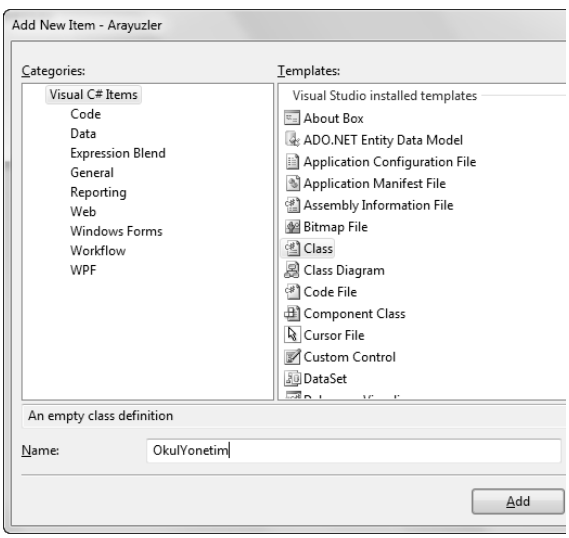
Artık Ogrenci sınıfından başka bir sınıf türetmenin imkânı kalmamıştır. Bu sözcük sınıfın hâlihazırda mühürlenip kapatıldığını belirtir.

UYGULAMA FAALİYETİ

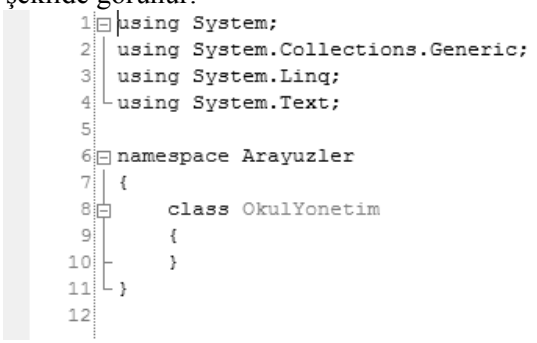
Arayüz (interface) ile çalışınız.

İşlem Basamakları	Öneriler
➢ Nesne tabanlı programlama yazılımı uygulamasını açınız.	➢ Başlat – Çalıştır seçeneğini tıklayınız, Çalıştır penceresi açılır. Aç kutusuna "devenv" yazıp Tamam düğmesine tıklayınız.
➢ Öğrenme Faaliyeti- 1'de oluşturduğunuz KalitimVeArayuzler çözümünü açın.	➢ Nesne tabanlı programlama yazılımını başlattığınızda varsayılan olarak Başlangıç Sayfasının (Start Page) görüntülenmesi gerekir. Bu sayfadaki Geçmiş Projeler (Recent Projects) bölümünden KalitimVeArayuzler çözümünü tıklayınız. ➢ Başlangıç sayfasını görüntülemek için Görünüm (View) menüsünden Diğer Pencereler (Other Windows) alt menüsü içinde yer alan Başlangıç Sayfasını (Start Page) tıklayabilirsiniz. 
➢ Çözüme yeni bir proje ekleyiniz.	➢ Dosya (File) menüsünden Ekle (Add) alt menüsü içindeki Yeni Proje… (New Project…) seçeneğini tıklayınız. Yeni Proje Ekle (Add New Project) penceresi açılır.

<table>
<tr>
<td></td>
<td> 

➢ Şablonlar (Templates) bölümünden Konsol Uygulamasını (Console Application) seçiniz. Ad (Name) kutusuna “Arayuzler” yazın ve Tamamı (OK) tıklayınız.</td>
</tr>
<tr>
<td>➢ Projeye yeni bir sınıf ekleyiniz.</td>
<td>➢ Proje (Project) menüsünden Sınıf Ekle… (Add Class…) seçeneğini tıklayınız. Dilerseniz Shift + Alt + C tuşlarına da basabilirsiniz. Yeni Öğe Ekle – Arayuzler (Add New Item - Arayuzler) penceresi görünür.


➢ Şablonlar (Templates) bölümünden Sınıfı (Class) seçiniz. Ad (Name) kutusuna “OkulYonetim” yazınız. Ekle (Add)</td>
</tr>
</table>

	düğmesine tıklayınız. Çözüm Gezgininde (Solution Explorer) "Arayuzler" projesine "OkulYonetim.cs" dosyası eklenir ve editör penceresinde "OkulYonetim.cs" sekmesi açılır.
➢ "OkulYonetim.cs" sekmesindeki kodu düzenleyiniz.	➢ "OkulYonetim.cs" kod editörü başlangıçta şu şekilde görünür:  ➢ Üstteki kodu aşağıdakiyle değiştiriniz. ➢ IInsan adında bir arayüz oluşturdunuz.
➢ IInsan arayüzünü Ogrenci ve Ogretmen sınıfları ile uygulayınız.	➢ Aşağıdaki kodları IInsan arayüzünün küme parantezlerinin altına ekleyiniz.

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

namespace Arayuzler
{
    class OkulYonetim
    {
    }
}
```

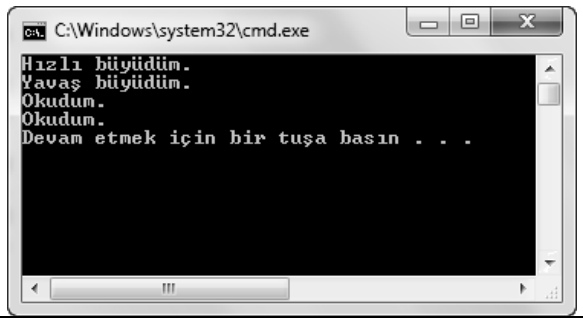
```
class OkulYonetim
{
}
```

```
interface IInsan
{
    void Buyu();
    void Oku();
}
```

```
class Ogrenci : IInsan
{
    public void Buyu()
    {
        Console.WriteLine("Hızlı
büyüdüm.");
    }

    public void Oku()
    {
        Console.WriteLine("Okudum.");
    }
}

class Ogretmen : IInsan
{
```

	`` public void Buyu() { Console.WriteLine("Yavaş büyüdüm."); } public void Oku() { Console.WriteLine("Okudum."); } } ``` ➢ IInsan arayüzünün Buyu() ve Oku() metotlarını Ogrenci ve Ogretmen sınıflarında uygulamış oldunuz. Bu sınıflar eğer IInsan arayüzünü eksik uygularlarsa derleme zamanı hatası alırsınız ve projeniz derlenmez. IInsan arayüzü Ogrenci ve Ogretmen sınıfları için birer sözleşme niteliğindedir ve bu sözleşmeye tamamen uymak zorundadır.
➢ Soyut (abstract) bir sınıf ekleyiniz.	➢ Ogrenci ve Ogretmen sınıflarında Oku metotlarının aynı olduklarını görüyorsunuz. Kod tekrarı oluştu! Bunu önlemek için OkuyanInsan soyut sınıfını oluşturup Oku metodunu bu sınıf içine alalım. ``` abstract class OkuyanInsan { public void Oku() { Console.WriteLine("Okudum."); } } ```
➢ IInsan arayüzü ile Ogrenci ve Ogretmen sınıflarından Oku metodunun tanımını ve uygulamalarını kaldırınız.	➢ IInsan arayüzü ile Ogretmen ve Ogrenci sınıflarını aşağıdaki şekilde yeniden yazınız. ``` interface IInsan { void Buyu(); } class Ogrenci : IInsan { public void Buyu() { Console.WriteLine("Hızlı büyüdüm."); } } ```

	`` class Ogretmen : IInsan { public void Buyu() { Console.WriteLine("Yavaş büyüdüm."); } } ```
➢ Ogrenci ve Ogretmen sınıflarını ortak uygulamayı OkuyanInsan sınıfından kalıtım yoluyla alınız.	➢ Ogrenci ve Ogretmen sınıflarını aşağıdaki şekilde yeniden yazınız. ``` class Ogrenci : OkuyanInsan, IInsan { public void Buyu() { Console.WriteLine("Hızlı büyüdüm."); } } class Ogretmen : OkuyanInsan, IInsan { public void Buyu() { Console.WriteLine("Yavaş büyüdüm."); } } ```
➢ Ogrenci ve Ogretmen sınıflarının örneklerini oluşturup uygulamanızı test ediniz.	➢ Program.cs dosyasındaki Main metodunu aşağıdaki şekilde yeniden yazınız. ``` static void Main(string[] args) { Ogrenci birOgrenci = new Ogrenci(); Ogretmen birOgretmen = new Ogretmen(); birOgrenci.Buyu(); birOgretmen.Buyu(); birOgrenci.Oku(); birOgretmen.Oku(); } ```
➢ Uygulamanızı test ederek sonuçlarını izleyiniz.	➢ Çözümünüz içinde iki proje olduğu için hangisinin çalışma sırasında giriş projesi olacağını belirlemeniz gerekir. Çözüm Gezgininde (Solution Explorer) “Arayuzler”

	projenizi sağ tıklayın. Açılan içerik menüsünden Başlangıç Projesi Yap (Set as StartUp Project) seçeneğini tıklayınız. ➢ Hata Ayıkla (Debug) menüsünden Hata Ayıklamadan Başlat (Start Without Debugging) seçeneğini tıklayınız. Dilerseniz alternatif olarak Ctrl + F5 tuşlarına da basabilirsiniz. Aşağıdaki görüntüyü görmeniz gerekir. 
➢ Nesne tabanlı programlama yazılımını kapatınız.	➢ Uygulama faaliyeti bitti. Dosya (File) menüsünden Çıkış (Exit) seçeneğini tıklayarak nesne tabanlı programlama yazılımını kapatınız.

KONTROL LİSTESİ

Bu faaliyet kapsamında aşağıda listelenen davranışlardan kazandığınız beceriler için **Evet,** kazanamadıklarınız için **Hayır** kutucuklarına (X) işareti koyarak öğrendiklerinizi kontrol ediniz.

Değerlendirme Ölçütleri	Evet	Hayır
1. Bir arayüz (interface) tanımlayabildiniz mi?		
2. Arayüz kısıtlarını sayabildiniz mi?		
3. Arayüzü uygulayabildiniz mi?		
4. Bir sınıfın örneğine arayüz değişkeni ile başvurabildiniz mi?		
5. Soyut (abstract) sınıf oluşturabildiniz mi?		
6. Mühürlenmiş (sealed) bir sınıf oluşturabildiniz mi?		

DEĞERLENDİRME

Değerlendirme sonunda **"Hayır"** şeklindeki cevaplarınızı bir daha gözden geçiriniz. Kendinizi yeterli görmüyorsanız öğrenme faaliyetini tekrar ediniz. Bütün cevaplarınız **"Evet"** ise "Ölçme ve Değerlendirme"ye geçiniz.

ÖLÇME VE DEĞERLENDİRME

Aşağıdaki soruları dikkatlice okuyunuz ve doğru seçeneği işaretleyiniz.

1. Bir arayüz tanımlamak için kullanılan sözcük aşağıdakilerden hangisidir?
A) enum
B) interface
C) class
D) struct

2. Bir arayüzün içerebileceği üye aşağıdakilerden hangisidir?
A) Değişken bildirimleri
B) Metot uygulamaları (deyim ve ifadeleri)
C) Sınıf tanımları
D) Metot tanımları

3. Aşağıdakilerden hangisi bir arayüz kısıtlaması değildir?
A) Arayüz içinde alan (field) tanımlanmaz.
B) Arayüz içinde kurucu (constructor) tanımlanmaz.
C) Arayüz içinde metot (method) tanımlanmaz.
D) Arayüz içinde yıkıcı (destructor) tanımlanmaz.

4. Bir arayüzü uygularken aşağıdakilerden hangisi bir kural veya alışkanlık olarak gerekli değildir?
A) Parametre adları arayüzde ve uygulayan sınıfta aynı olmalıdır.
B) Metot adları ve dönüş türleri kesin olarak eşleşmelidir.
C) Parametreler (ref ve out anahtar sözcükleri dahil) tam olarak eşleşmelidir.
D) Metot adı arayüz adıyla başlar.

5. IInsan adında bir arayüzünüz var. Bu arayüzü Ogrenci, Mudur ve Ogretmen adında üç sınıf uyguluyor olsun. birInsan adında IInsan değişkeni, birOgrenci adında Ogrenci değişkeni, birOgretmen adında Ogretmen değişkeni, birMudur adında da Mudur değişkeni bildirildiği varsayılsın. Buna göre aşağıdaki atamalardan hangisi geçerli değildir?
A) `birInsan = birMudur;`
B) `birInsan = birOgrenci;`
C) `birInsan = birOgretmen;`
D) `birMudur = birInsan;`

DEĞERLENDİRME

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

MODÜL DEĞERLENDİRME

Aşağıdaki soruları dikkatlice okuyunuz ve doğru seçeneği işaretleyiniz.

1. İnsan adında bir temel sınıf olsun. Ögrenci sınıfı da Insan sınıfından türetildi. Ogrenci sınıfı kurucuda parametre olarak öğrenci adını alsın. birOgrenci adında Insan türünde bir değişken bildirip Ogrenci sınıfından bir nesneyi birOgrenci değişkenine atanmak isteniyor. Aşağıdakilerden hangisi bu işi doğru bir şekilde gerçekleştirir?

A)
```
Ogrenci birOgrenci = new Ogrenci("Ali UZUN");
```
B)
```
Insan birOgrenci = new Insan("Ali UZUN");
```
C)
```
Ogrenci birOgrenci = new Insan("Ali UZUN");
```
D)
```
Insan birOgrenci = new Ogrenci("Ali UZUN");
```

2. Temel sınıfın sahip olduğu bir metodun imzası bakımından aynısına türemiş sınıf da sahip olursa derleyici bir uyarı (warning) görüntüler. Bu uyarıyı kaldırmak için kullanılan sözcük aşağıdakilerden hangisidir?

A) protected
B) virtual
C) new
D) override

3. Temel sınıfta sanal bir metot yazarak daha sonra türemiş sınıfta da bu metodu geçersiz kılınmak isteniyor. Bu sanal metodu tanımlamak için kullanılan sözcük aşağıdakilerden hangisidir?

A) protected
B) virtual
C) new
D) override

4. Temel sınıfta sanal bir metot yazarak daha sonra türemiş sınıfta da bu metodu geçersiz kılınmak isteniyor. Bu sanal metodu türemiş sınıfta geçersiz kılmak için kullanılan sözcük aşağıdakilerden hangisidir?

A) protected
B) virtual
C) new
D) override

5. Temel sınıfın bir üyesine, sadece bu sınıftan türemiş bir sınıfın erişebilmesi isteniyor. Bu üyeyi tanımlamak için kullanılabilecek erişim belirteci aşağıdakilerden hangisidir?

A) Ortak (public)
B) İç (internal)
C) Korumalı (protected)
D) Özel (private)

6. Bir sınıfı soyut tanımlamak için kullanılan sözcük aşağıdakilerden hangisidir?

A) abstract
B) virtual
C) static
D) sealed

7. Bir sınıfı kendisinden kalıtım yoluyla başka bir sınıf türetilemeyecek şekilde tanımlamak için kullanılan sözcük aşağıdakilerden hangisidir?
A) virtual
B) abstract
C) static
D) sealed

8. Arayüz adlarını "I" harfiyle başlatmanın yararı aşağıdakilerden hangisi olabilir?
A) Derleyici bunun bir arayüz olduğunu anlar.
B) Programcılar arasında uyum sağlanır.
C) Bir sınıfı bir arayüzden kesin olarak ayırır.
D) Bir yapıyı bir arayüzden kesin olarak ayırır.

9. Arayüzde tanımlanan metotlarda erişim belirteci belirtilmemesinin nedeni aşağıdakilerden hangisi olabilir?
A) Bir arayüzdeki tüm metotlar dolaylı olarak ortaktır.
B) Bir arayüzdeki tüm metotlar dolaylı olarak özeldir.
C) Bir arayüzdeki tüm metotlar dolaylı olarak korumalıdır.
D) Bir arayüzdeki tüm metotlar tam sayı değer döndürür.

10. Bir arayüzün içerebileceği tür aşağıdakilerden hangisidir?
A) Sınıf
B) Yapı
C) Numaralandırma
D) Yukarıdakilerin hiçbiri

11. Bütün üyeleri soyut olarak işaretlenmiş bir soyut sınıf aşağıdakilerden hangisine benzer?
A) Protected olarak işaretlenmiş üyelere
B) Tüm üyeleri virtual sözcüğü ile işaretlenmiş temel sınıflara
C) Arayüzler
D) Yukarıdakilerin hiçbiri

DEĞERLENDİRME

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

CEVAP ANAHTARLARI

ÖĞRENME FAALİYETİ-1'İN CEVAP ANAHTARI

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

ÖĞRENME FAALİYETİ-2'İN CEVAP ANAHTARI

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

MODÜL DEĞERLENDİRMENİN CEVAP ANAHTARI

1	**D**
2	**C**
3	**B**
4	**D**
5	**C**
6	**A**
7	**D**
8	**B**
9	**A**
10	**D**
11	**C**

KAYNAKÇA

- TEZCAN Ümit(çeviren)**, Adım Adım Microsoft C# 2008,** Arkadaş Yayınevi, Ankara, 2009 (Orijinal Kaynak: SHARP John, Microsoft C# 2008 Step By Step, Microsoft Press, Redmond, 2008).