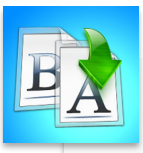
ERİŞİM BELİRLEYİCİLER



İÇİNDEKİLER

- Public
- Private
- Protected
- Internal
- Protected Internal

HEDEFLER

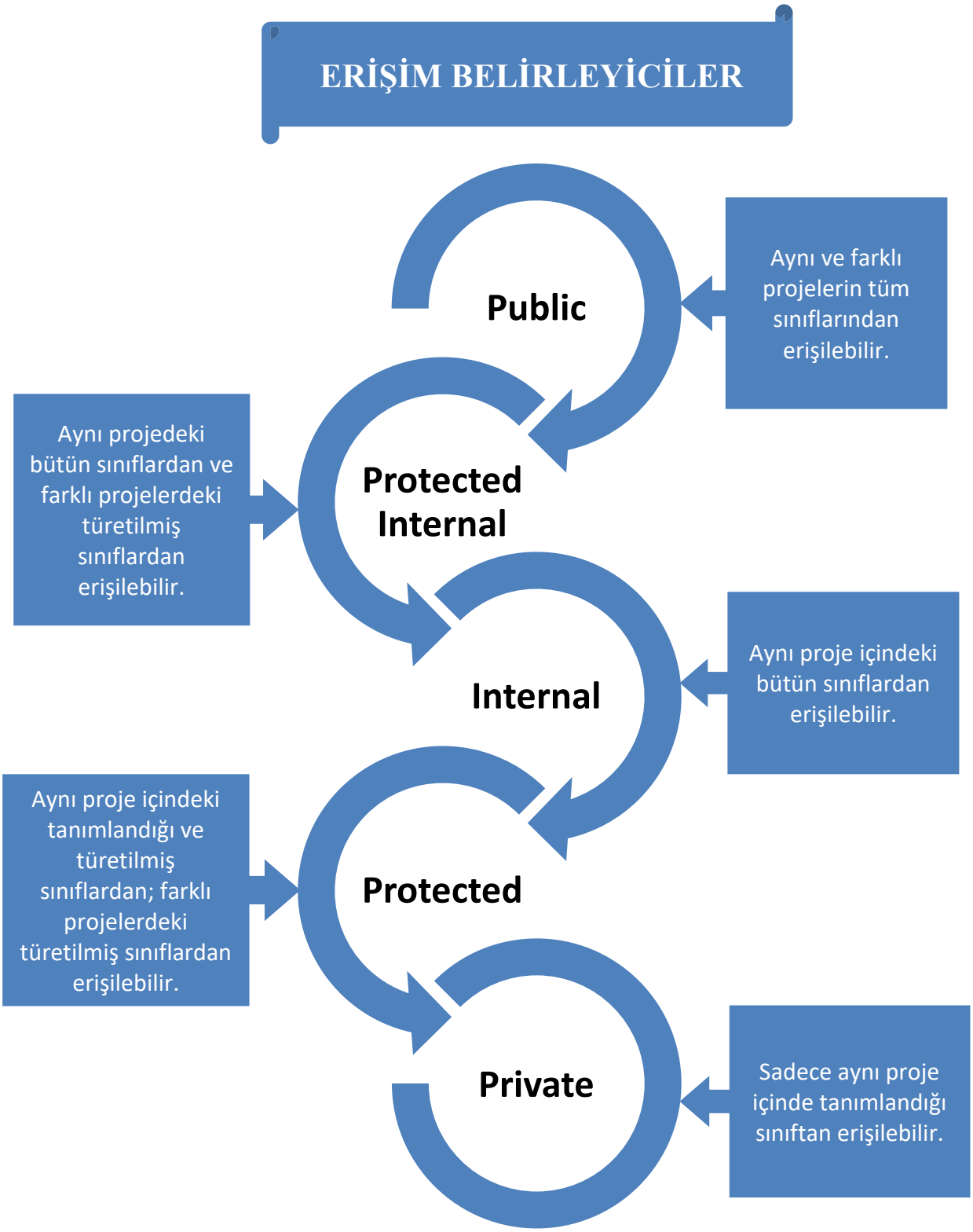
- Bu üniteyi çalıştıktan sonra;
  - Erişim belirleyicilerin ne amaçla kullanıldığını anlayabilecek,
  - Aynı ve farklı projelerde erişim belirleyicilerinin etkilerini görebilecek,
  - Sınıfların, özelliklerin ve metotların erişim seçeneklerini belirleyebileceksiniz.

Atatürk Üniversitesi
Açıköğretim Fakültesi

NESNE TABANLI PROGRAMLAMA I

Öğr. Gör.
Daha ORHAN

ÜNİTE
7


ERİŞİM BELİRLEYİCİLER
Public
Aynı ve farklı projelerin tüm sınıflarından erişilebilir.
Protected Internal
Aynı projedeki bütün sınıflardan ve farklı projelerdeki türetilmiş sınıflardan erişilebilir.
Internal
Aynı proje içindeki bütün sınıflardan erişilebilir.
Protected
Aynı proje içindeki tanımlandığı ve türetilmiş sınıflardan; farklı projelerdeki türetilmiş sınıflardan erişilebilir.
Private
Sadece aynı proje içinde tanımlandığı sınıftan erişilebilir.

GİRİŞ

C# programlama dili nesne yönelimli programlama dilleri arasında yer almaktadır. Bununla birlikte birçok programlama diline nesne yönelimli programlama anlayışını kazandırma noktasında öncülük ettiği söylenebilir. Burada diğer programlama dillerinin birbirlerine göre kıyaslanmasından ziyade nesne yönelimli programlama anlayışının avantajlarından bahsetmenin faydalı olacağı düşünülmektedir.

Bir nesne ile diğer nesneler arasında ilişki oluşturmamıza olanak sağlayan nesne yönelimli programlama anlayışının birçok avantajı bulunmaktadır. Nesne yönelimli programlamanın en büyük özelliklerinden biri olan sınıf yapısı ile nesneler türeterek bu nesneler üzerinde değişiklik yapmamıza olanak sağlanmaktadır. Böylelikle yapılmak istenen değişiklikleri programın tamamında tek tek yapmak yerine oluşturulan nesne üzerinde bir defa yapmak yeterli olacaktır.

Nesne yönelimli programlama hata ayıklama sürecinde de işimizi fazlasıyla kolaylaştırmaktadır. Örneğin herhangi bir x değişkeninin istenmedik değerler aldığını varsayalım. Hazırladığımız programda bu x değeri birçok yerde bulunacak ve programın herhangi bir öğesi bu değer ile uğraşıyor olabilecek. Blok yapısının olmadığı ve kapsüllemenin bulunmadığı bir programlama yaklaşımında çok büyük kod satırları içinden hatalı olan değeri bulmak neredeyse imkânsız gibi bir hale geliyor. Ayrıca hatayı bulmak yeterli olmayabilir çünkü bu değeri programın her alanında düzeltmeniz gerekiyor. İşte bu noktada da nesne yönelimli programlama yaklaşımı bize çok büyük avantaj sağlamaktadır. Hatanın özüne inmek ve hatanın düzeltilmesiyle tüm programın düzene girmesi çok hızlı bir şekilde gerçekleştirilebilmektedir.

Nesne yönelimli programlama anlayışı hata ayıklama sürecinde bize kolaylıklar sağlamaktadır.

Oluşturulan sınıflar sayesinde daha az kod yazılmakta ve daha fazla iş yapılabilmektedir. Aynı zaman da oluşturulan sınıflar sayesinde programcıların kod tekrarından kurtulmasını sağlamaktadır. Bu sayede yazılım geliştirme süreci hızlanmakta ve verimlilikte artmaktadır. Ayrıca sınıflar içerisinde oluşturulan nesneler diğer projelerde de rahat bir şekilde kullanılabilmektedir.

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

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

internal erişim belirleyicisi anlatılacak olup private protected erişim belirleyicisi bireysel etkinlik olarak sizlere bırakılmıştır. Bu ünite kapsamında anlatılacak olan erişim belirleyicileri aşağıda maddeler halinde sunulmuştur.

- Public
- Private
- Internal
- Protected
- Protected internal

Bu erişim belirleyicilerinin her biri farklı amaçlarla kullanılmaktadır. Erişim belirleyicilerini anlatmaya başlamadan önce genel bir bilgilendirme yapmanın faydalı olacağı düşünülmektedir. Erişim belirleyicileri kısıtlarını ve yetilerini iki başlık altında ele almak gerekir. Bunlar tek bir proje içinden erişim ve farklı projelerden erişim olacak şekildedir.

Tek bir proje içindeki erişimi üç başlık altında toplayabiliriz. Bu başlıklar;

- Tanımlandığı sınıftan
- Türetilmiş sınıflardan
- Diğer sınıflardan

Bütün erişim belirleyiciler tanımlandığı sınıftan erişim sağlanmasına izin vermektedir. Türetilmiş sınıflardan ve diğer sınıflardan erişim yapılmasını engellemek için kullanabileceğimiz sadece private erişim belirleyicisidir. Son olarak aynı proje içinden diğer sınıflardan erişimi engellemek için private ve protected erişim belirleyicilerini kullanabilmekteyiz.

Bütün erişim belirleyicileri tanımlandığı sınıftan erişim sağlanmasına izin vermektedir.

Bunun haricinde farklı projeler içinden erişim durumu iki başlık altında incelenebilir. Bunlar;

- Diğer sınıflardan
- Türetilmiş sınıflardan

Diğer sınıflara erişim sağlayabilmek için sadece public erişim belirleyicisini kullanabiliriz. Farklı projelerde bulunan türetilmiş sınıflara erişim izni vermek için ise public, protected ve protected internal erişim belirleyicileri ile tanımlamalarımızı gerçekleştirebiliriz.

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

- Namespace ve Enum, erişim belirleyiciler ile nitelendirilemezler. Çünkü daimi olarak public tanımlıdırlar.
- Sınıflar protected ya da private olarak nitelendirilemezler. Public veya internal olarak tanımlanabilmektedirler.
- Türetilen sınıflarda türetildikleri sınıfların erişim kısıtlamaları geçerlidir. Kalıtım yoluyla alınan bu kısıtlamalar artırılabilir ancak azaltılamazlar.
- Programlama dillerinin sözdizimine bağlı olarak erişim belirleyicilerle yapılan tanımlamalarda büyük küçük harf duyarlılığına dikkat edilmelidir.

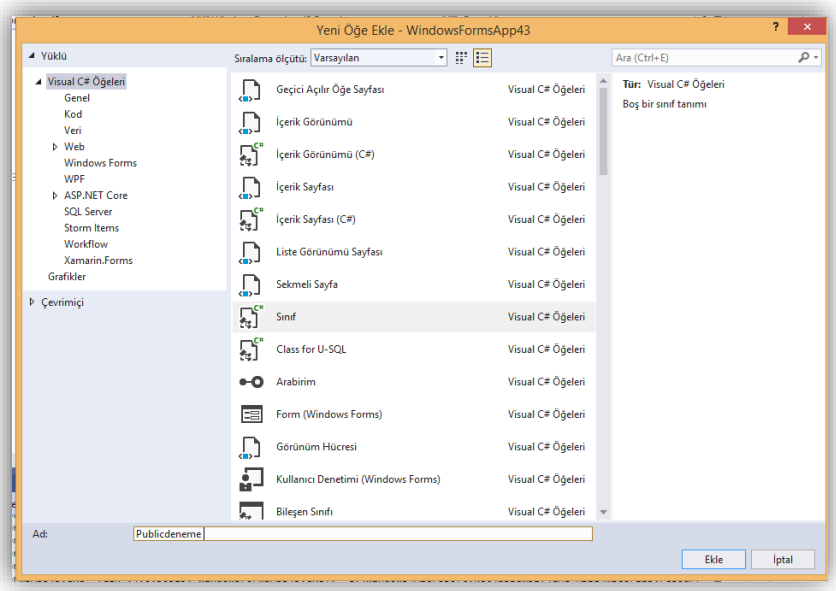
Public

Oluşturduğumuz metotların ve özelliklerin erişilebilirliğinin ayarlanabileceğinden bahsetmiştik. *Public erişim belirleyicisini kullandığımızda, kısıtlama olmayan yani sınıf içi, sınıf dışı ve farklı projeler olmak üzere her yerden erişilebilir olmasını sağlamaktayız.* Tabi public olarak tanımlanan değişkenin ya da metodun farklı projelerde (exe dosyalarında) kullanılabilmesi için başvuru olarak eklenmesi gerekmektedir.

Erişim belirleyicileri genel olarak tek bir sayfa üzerinde anlatılabilecek kadar kolay olsa da bunların kullanımı örnekler üzerinde gösterilecektir. Örneklendirmeler adım adım anlatılacağı için sizlerin de bu örnekleri uygulaması konuyu tam olarak anlamanız noktasında büyük bir önem arz etmektedir.

Öncelikle public olarak tanımlanan bir özelliğe başka bir sınıf üzerinden nasıl erişilebildiğini göstereceğimiz bir uygulama tasarlayalım. Visual Studio 2019 programını başlattıktan sonra C# programlama dilini seçerek Windows forms uygulamasını başlatalım. Açılan pencerede menü çubuğundan bulunan *Proje* menüsünü işaretleyerek *Sınıf Ekle…* komutunu verelim. Bu işlemin ardından Şekil 7.1.'de gösterildiği gibi *Yeni Öğe Ekle* penceresi karşımıza gelecektir. Bu pencerede *Sınıf* seçeneğini işaretleyip *Ad* alanına *Publicdeneme* yazarak *Ekle* butonuna tıklayalım.

Public erişim belirleyicisi hiçbir sınırlama getirmemektedir.



Şekil 7.1. Yeni Öğe Ekle Penceresi

Publicdeneme sınıfı içinde *string* veri türünde değer alabilecek *isim* özelliğini *public* olarak tanımlayalım. Bu işlem kodları aşağıda gösterilmektedir.

```
class publicdeneme
{
  public string isim;
}
```

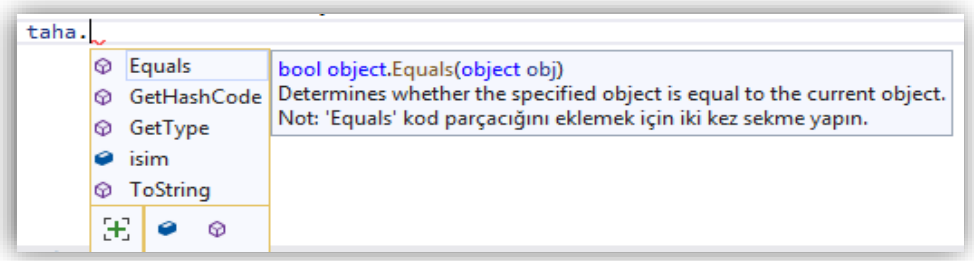
Böylelikle herhangi bir kısıtlaması olmayan isim adında bir özellik tanımlamış olduk. Şimdi tasarım alanımıza dönerek F7 fonksiyon tuşu ile kod editörünü açalım. Bu işlem menü çubuğunda bulunan *Görünüm* menüsündeki *Kod* komutuyla da gerçekleştirilebilmektedir.

Burada Taha adında bir nesne türeterek publicdeneme sınıfında oluşturduğumuz özelliğin Taha nesnesinde kullanılabileceğini göreceğiz. Nesne türetme işlemi detaylı olarak önceki bölümlerde anlatıldığı için burada üzerinde durulmayacaktır. Kod editörünü aşağıdaki gibi düzenleyelim.

```
public Form1()
{
  InitializeComponent();
  publicdeneme taha = new publicdeneme();
  taha.isim = "tahaorhan";
}
```

Kodlarımızı yazarken *Taha* nesnesini yazıp nokta (.) koyduktan sonra özellikler listesinde Şekil 7.2.'de gösterildiği gibi *isim* özelliğinin de yer aldığını göreceksiniz. C# programlama dilinde nokta kullanımı ile açılan bu liste kullanımı *IntelliSense* olarak adlandırılmaktadır.

Tasarım alanında çalışırken F7 fonksiyon tuşu ile kod editörüne geçilebilir.

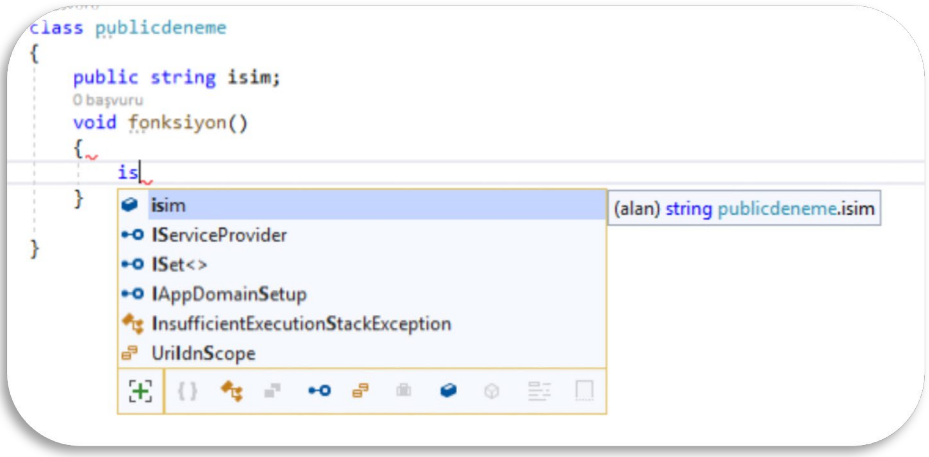


Şekil 7.2. Özellikler Listesi İsim Özelliği

Public olarak tanımlanan bir özelliğe kendi sınıfı içinden de ulaşılabilmektedir. Bu durumu basit bir metot tanımlayarak gösterebiliriz. Publicdeneme isimli sınıfımız içine aşağıdaki kodları yazalım.

```
class publicdeneme
{
  public string isim;
  void fonksiyon()
  {
        isim = "taha orhan";
  }
}
```

Görüldüğü üzere kodlarımızı yazarken public olarak tanımladığımız özellik kendi sınıfı içinde listelenmektedir. Bu duruma ait ekran görüntüsü Şekil 7.3.'de verilmiştir.



Şekil 7.3. Publicdeneme Sınıfına Ait Metot Özellikler Listesi

Bireysel Etkinlik

- Bir sınıf oluşturarak string veri türünde public erişim belirleyicisi ile bir özellik tanımlayın. Bu özelliğe girilen değeri hazırlayacağınız metot yardımıyla büyük harflerle kullanıcıya mesaj yoluyla gösterin. (Not: Harfler ToUpper yöntemi ile büyük harfe dönüştürülebilir.)

Erişim belirleyicileri içinde en kısıtlayıcı olan erişim belirleyicisi private erişim belirleyicisidir.

Private

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

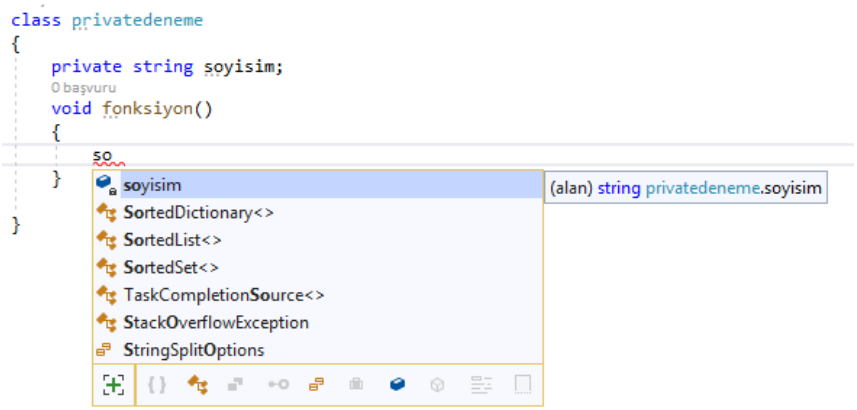
Öncelikle public erişim belirleyicisinde yapmış olduğumuz uygulamaya benzer bir uygulama geliştirerek private erişim belirleyicisini basit bir şekilde anlatalım. Ardından hem public hem de private erişim belirleyicilerini beraber anlatacağımız daha somut bir uygulama geliştireceğiz.

Yeni bir sınıf ekleyelim ve bu sınıfın ismi privatedeneme olsun. Bu sınıf içinde *soyisim* adında bir özellik tanımlayıp bu özelliğe sadece privatedeneme sınıfından erişilebildiğini gözlemleyeceğiz. Sınıfımızı oluşturduktan sonra kodlarımızı aşağıdaki gibi düzenleyelim.

```
class privatedeneme
{
```

```
    private string soyisim;
    void fonksiyon()
    {
        soyisim = "orhan";
    }
}
```

Kodlarımızı yazarken soyisim özelliğinin privatedeneme sınıfı içinde oluşturulan fonksiyon metodunda herhangi bir hata mesajı alınmadan kullanılabildiğini gözlemledik. Aynı zamanda bu özelliğin liste halinde sunulması Şekil 7.4.'de gösterilmektedir.



Şekil 7.4. Private Erişim Belirleyicisinin Kendi Sınıfında Kullanılması

Private erişim belirleyicisi kullanılarak oluşturulan bir özelliğe farklı sınıflardan erişilemeyeceğini söyledik. Public erişim belirleyicisinde olduğu gibi farklı bir sınıf içinde yazmak istersek çalıştırma aşamasına geçmeden kod editöründe kırmızı bir alt çizgi ile program bizi uyarmaktadır. Bu duruma ait ekran görüntüsü Şekil 7.5.'de gösterilmektedir.

Private erişim belirleyicisi kullanılarak oluşturulan bir özelliğe farklı sınıflardan erişilememektedir.

```
public partial class Form1 : Form
{
    1 başvuru
    public Form1()
    {
        InitializeComponent();
        publicdeneme taha = new publicdeneme();
        taha.isim = "tahaorhan";
        privatedeneme orhan = new privatedeneme();
        orhan.soyisim = "Orhan";
    }
}
```

Şekil 7.5. Private tanımlanan özellikler başka sınıflarda kullanılamaz

Şimdi public ve private erişim belirleyicilerimizi birlikte kullanacağımız daha somut bir uygulama geliştirelim. Uygulamamıza *arac* isimli bir sınıf ekleyelim. Bildiğiniz üzere araçlar üretilirken farklı model ve kasa yapıları kullanıcılara sunulmaktadır. Bu konudaki tercih tamamen kişisel isteklere bağlı olmaktadır.

Ancak her araç üretilirken fabrika çıkışlı olarak şasi numarası belirlenmektedir. Bu numara insanların vatandaşlık numarası gibi tek ve özeldir. Şasi numarası üretim aşamasında belirlenir ve değiştirilemez bir değere sahiptir. Bizde uygulamamızı bu doğrultuda tasarlayalım. Buna göre araç sınıfımızda model, kasa ve sasi olmak üzere üç özellik tanımlayalım. Bu özelliklerden model ve kasa herkesin erişebileceği public erişim belirleyicisiyle sasi ise sadece tanımlandığı sınıf üzerinden değiştirilebilecek şekilde olan private erişim belirleyicisi ile oluşturulacaktır. Bu doğrultuda araç sınıfımıza ait kodları aşağıdaki gibi hazırlayalım.

```
class arac
  {
    public string model;
    public string kasa;
    private string sasi;
    void fonk()
    {
      model = "segment";
      kasa = "araç kasa yapısı";
      sasi = "fabrika ayarı";
    }
  }
```

Arac sınıfımızın kodlarını yazarken bütün özelliklere *fonk* metodu içinde erişilebildiğini gözlemlemişsinizdir. Şimdi Windows form uygulamamızın kod editörüne geçerek arac sınıfına ait iki adet nesne oluşturalım. Bu nesnelerin public olarak tanımlanan model ve kasa özelliklerini aşağıda verilen kod satırlarında olduğu gibi düzenleyelim.

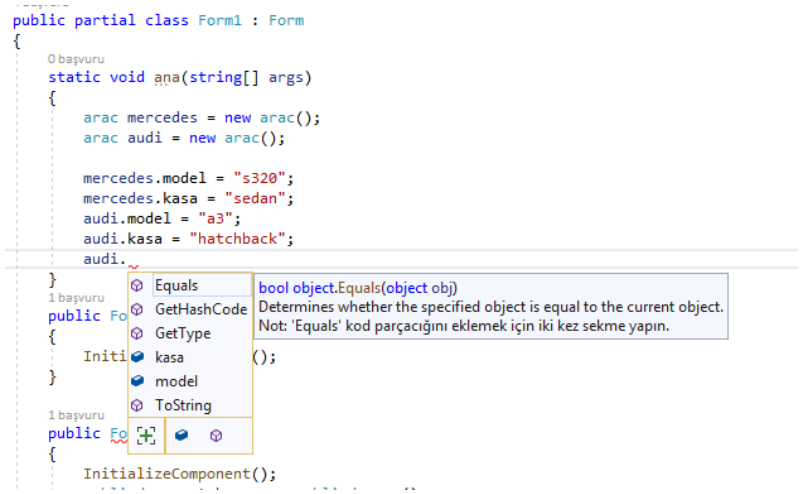
```
public partial class Form1 : Form
  {
    static void ana(string[] args)
    {
      arac mercedes = new arac();
      arac audi = new arac();

      mercedes.model = "s320";
      mercedes.kasa = "sedan";
      audi.model = "a3";
```

Private erişim belirleyicisi kullanılarak oluşturulan bir özelliğe farklı projelerden erişilememektedir.

```
            audi.kasa = "hatchback";
        }
        public Form1()
        {
            InitializeComponent();
        }
    }
```

Kodlarımızı yazarken model ve kasa özellikleri herhangi bir kısıtlama olmadan değiştirilmiştir. Ancak sasi özelliği farklı bir sınıf içinde bulunulduğu için erişime kapalıdır. Ana form içinde düzenlediğimiz nesnelerin özelliklerine yönelik ekran görüntüsü Şekil 7.6.'da olduğu gibi karşımıza gelmektedir.



Şekil 7.6. Public Tanımlanan Özelliklerin Listede Bulunmasına Yönelik Ekran Görüntüsü

Protected

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

Protected erişim belirleyicisi kullanılarak tanımlanan özellik ve metotlara aynı sınıf içinden erişilebilmektedir.

Protected olarak tanımlanan özellik ve metotlara o sınıftan türetilmiş sınıflardan da erişebilmek mümkündür. Private erişim belirleyicisinde böyle bir durum söz konusu değildir. Bu farkı gözlemleyebileceğimiz bir uygulama geliştirelim. Öncelikle projemize bir adet arac isimli bir sınıf ekleyelim. Bu sınıf içinde model, kasa ve sasi olmak üzere farklı erişim belirleyicilerini kullanabileceğimiz üç adet özellik tanımlayalım. Bu özelliklerden model public, kasa protected ve sasi private olarak tanımlanmış olsun. Bu üç özelliğe aynı sınıf üzerinde tanımlayacağımız fonk isimli metot üzerinden erişilebilmektedir.

Projemizin kodlarını aşağıdaki gibi düzenlediğimiz zaman hiçbir hata mesajı ile karşılaşmayacağımızı fark edeceksiniz.

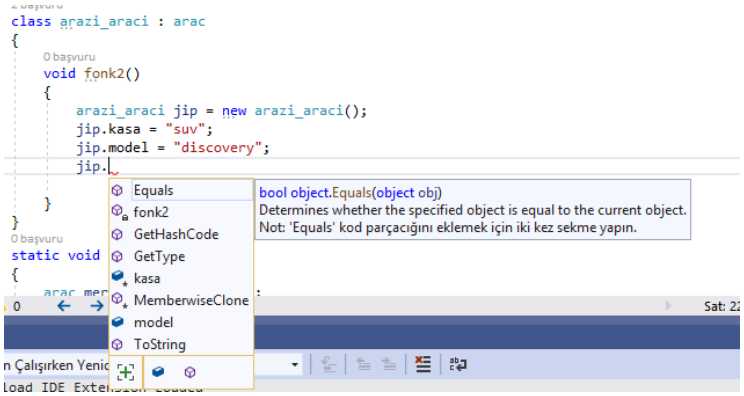
```
class arac
  {
    public string model;
    protected string kasa;
    private string sasi;
    void fonk()
    {
      model = "segment";
      kasa = "araç kasa yapısı";
      sasi = "fabrika ayarı";
    }
  }
```

Şimdi protected erişim belirleyicisinin private erişim belirleyicisinden farkını görebileceğimiz arac sınıfından *arazi_araclari* isimli bir sınıf türetelim. Ardından türetilen sınıf içerisinde *fonk2* isimli bir metot oluşturup *jip* nesnesini türetelim. Burada public olarak tanımladığımız model özelliğine daha önce yaptığımız uygulamalardan dolayı erişebileceğimizi biliyoruz. Protected olarak tanımlanan özellik de hatasız bir şekilde kullanılabilmektedir. Aşağıda bulunan kodları projemize ekleyelim.

```
class arazi_araci : arac
  {
    void fonk2()
    {
      arazi_araci jip = new arazi_araci();
      jip.kasa = "suv";
      jip.model = "discovery";
    }
  }
```

Protected erişim belirleyicisi kullanılarak tanımlanan özellik ve metotlara farklı sınıf içinden erişilememektedir.

Kodlarımızı yazarken sasi özelliğinin jip nesnesinde kullanılamayacağı anlaşılmaktadır. Bununla birlikte jip nesnemizi yazdıktan sonra nokta koyduğumuzda özellikler listesinde sadece model ve kasa özelliklerinin bulunduğu görülmektedir. Bu duruma ait ekran görüntüsü Şekil 7.7.'de gösterilmektedir.

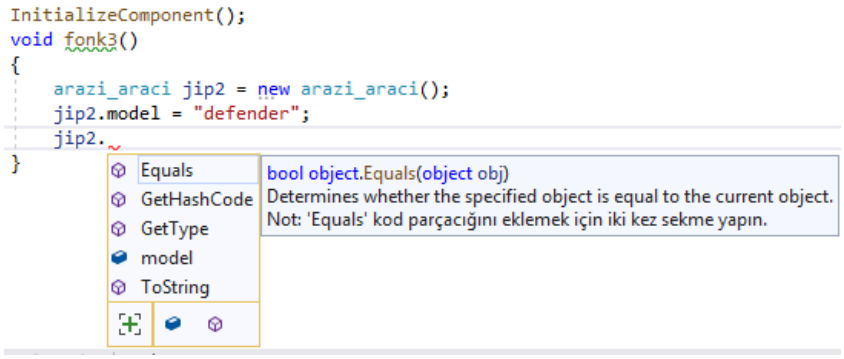


Şekil 7.7. Jip Nesnesine Ait Özellikler Listesi

Bu uygulamanın ardından private ve protected erişim belirleyicilerinin arasındaki fark anlaşılmış olacaktır. Şimdi protected ve public erişim belirleyicilerinin arasındaki farkı görebilmek için projemize devam edelim. Projemizin kod editörüne geçerek fonk3 isimli bir metot oluşturup bu metot içinde jip2 isimli arazi_arac sınıfına ait bir nesne türetelim. Kodlarımız aşağıdaki gibi olacaktır.

```
public partial class Form1 : Form
  {
    public Form1()
    {
      InitializeComponent();
      void fonk3()
      {
        arazi_araci jip2 = new arazi_araci();
        jip2.model = "defender";
      }
    }
  }
```

Dikkat ederseniz farklı bir sınıf üzerinde çalışıldığı için protected olarak tanımlanan kasa özelliği eklenememiştir. Kodlarımızı yazarken jip2 nesnesinin ardından nokta koyduğumuz zaman özellikler listesinde sadece model özelliğinin olduğu da fark edilmektedir. Bu durum Şekil 7.8.'de gösterilmektedir.



Şekil 7.8. Jip2 Nesnesine Ait Özellikler Listesi

Tek bir proje üzerinde çalışırken internal erişim belirleyicisi ile public erişim belirleyicisi arasında bir fark bulunmamaktadır.

Internal

Tek bir proje üzerinde çalışırken internal erişim belirleyicisi ile public erişim belirleyicisi arasında bir fark bulunmamaktadır. Internal erişim belirleyicisi kullanılarak tanımlanan özellik ve metotlara aynı sınıftan, türetilen sınıflardan ve farklı sınıflardan erişim mümkündür. Buradaki fark iki projenin tek bir çözüm gezgini üzerinde çalışırken ortaya çıkmaktadır.

Öncelikle tek bir proje üzerinde çalışırken internal ve public olarak tanımlanan iki özelliğin de aynı şekilde kullanılabileceğini gösteren bir uygulama geliştirelim. Visual Studio 2019 programında C# programlama dili ile başlattığımız Windows form uygulamasında *benzer* isimli bir sınıf oluşturalım. Bu sınıf içerisinde string veri türünde public olarak tanımlanan isim özelliği ve yine string veri türünde ancak erişim belirleyicisi olarak internal tanımlanan soyisim özelliğini kullanalım. Bu iki özelliğe aynı sınıf içinde erişebileceğimiz fonk isimli metodumuz olsun. Kodlarımızı aşağıdaki gibi düzenleyelim.

```
class benzer
  {
     public string isim;
     internal string soyisim;
     void fonk()
     {
        isim = "taha";
        soyisim = "orhan";
     }
  }
```

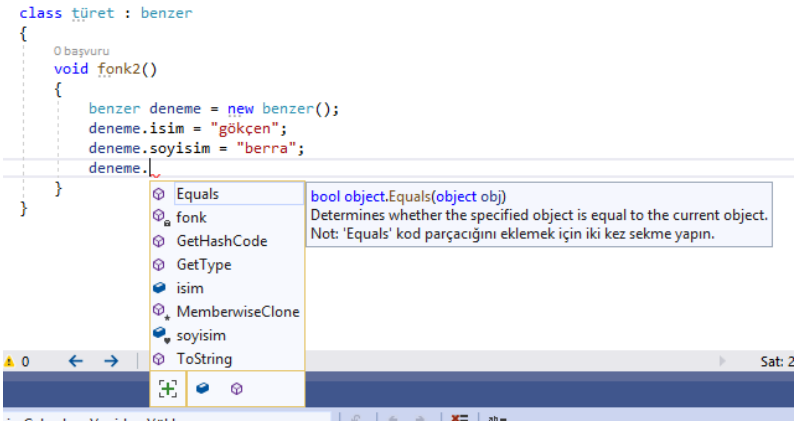
Görüldüğü üzere aynı sınıf içinde public ve internal olarak tanımlanan her iki özelliğe de erişilebilmektedir. Ek olarak benzer sınıfını kalıtım yoluyla alacak *türet* sınıfımızı oluşturalım. Bu sınıf içerisinde fonk2 isimli bir metot oluşturup deneme isimli bir nesne üzerinde public ve internal olarak tanımlanan özelliklerin

erişilebilirliğini denetleyelim. Bu işlem için benzer sınıfının altına aşağıdaki kodları ekleyelim.

```
class türet : benzer
    {
      void fonk2()
      {
        benzer deneme = new benzer();
        deneme.isim = "gökçen";
        deneme.soyisim = "berra";
      }
    }
```

Türetilen sınıflarda internal ve public olarak tanımlanan özellikler arasında bir fark yoktur.

Türetilen sınıflarda da internal ve public olarak tanımlanan özellikler arasında bir fark olmadığı anlaşılmaktadır. Bu durum Şekil 7.9.'da deneme nesnesinin özellikler listesinde de görülmektedir.



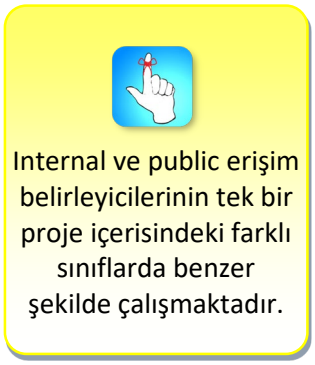
Şekil 7.9. Deneme İsimli Nesnenin Özellikler Listesi

Tek bir proje için farklı sınıflarda internal ve public erişim belirleyicileri arasında bir fark olup olmadığını kontrol edelim. Bu işlem için projemizin kod editörüne geçerek fonk3 isimli metot içerisinde deneme2 isimli nesnemizi türetip kodlarımızı aşağıdaki gibi düzenleyelim.

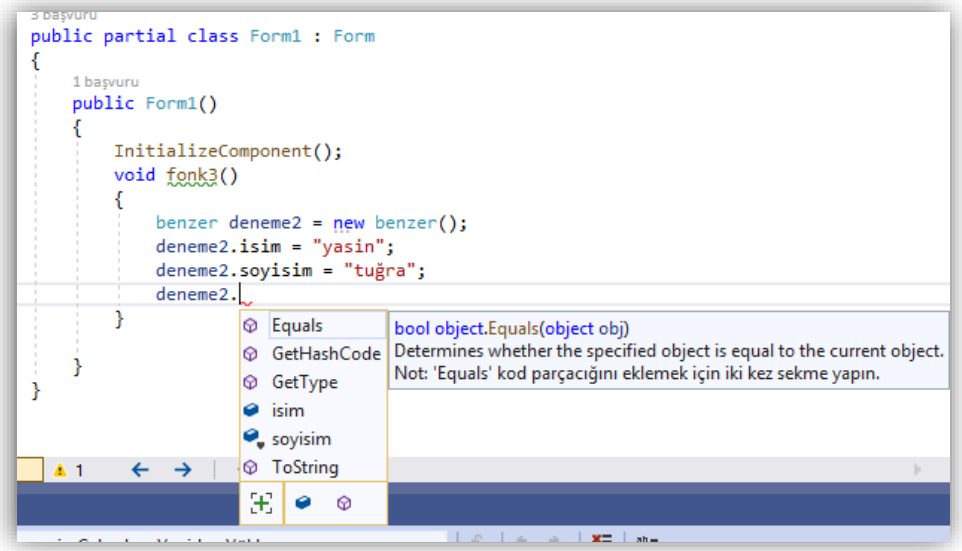
```
public partial class Form1 : Form
  {
    public Form1()
    {
      InitializeComponent();
      void fonk3 ()
      {
        benzer deneme2 = new benzer();
```

```
            deneme2.isim = "yasin";
            deneme2.soyisim = "tuğra";
        }
    }
}
```

Kodlarımızı yazdığımız zaman internal ve public erişim belirleyicilerinin tek bir proje içerisindeki farklı sınıflarda da benzer şekilde çalıştığı anlaşılmaktadır. Bu durum Şekil 7.10.'da deneme2 nesnesinin özellikler listesinde de görülmektedir.

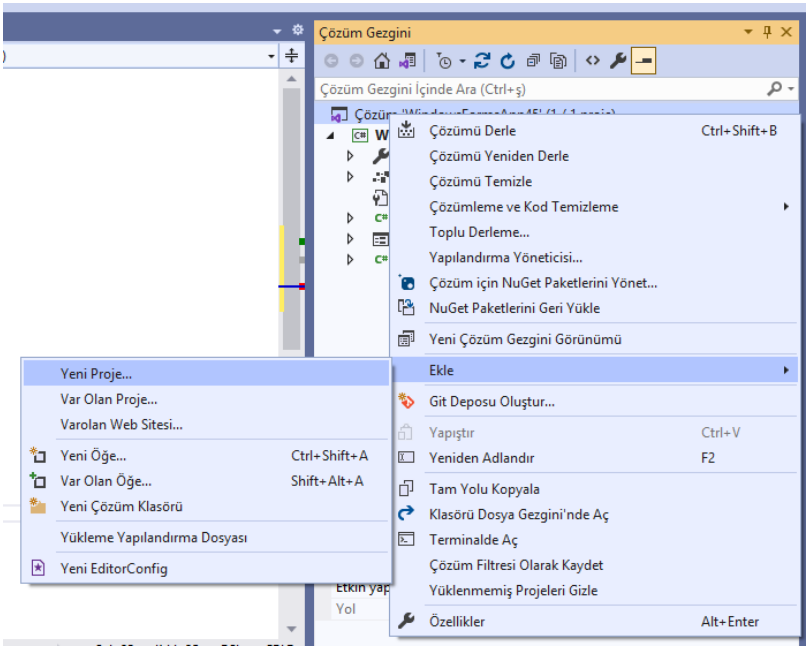


Internal ve public erişim belirleyicilerinin tek bir proje içerisindeki farklı sınıflarda benzer şekilde çalışmaktadır.



Şekil 7.10. Deneme2 İsimli Nesnenin Özellikler Listesi

Şimdi internal ve public erişim belirleyicilerinin farkını ortaya koyabilecek bir uygulama geliştirelim. Bu uygulama da farklı iki projenin tek bir çözüm üzerinde çalıştırılması söz konusu olduğu için yapılacak işlemler en basit haliyle düzenlenmiştir. Hazırladığımız uygulamanın çözüm gezgini üzerinde bulunan proje ismine farenin sağ tuşuna basıp Şekil 7.11.'de gösterildiği gibi *Ekle* -> *Yeni Proje…* komutunu vererek yeni bir proje ekle penceresini açalım.

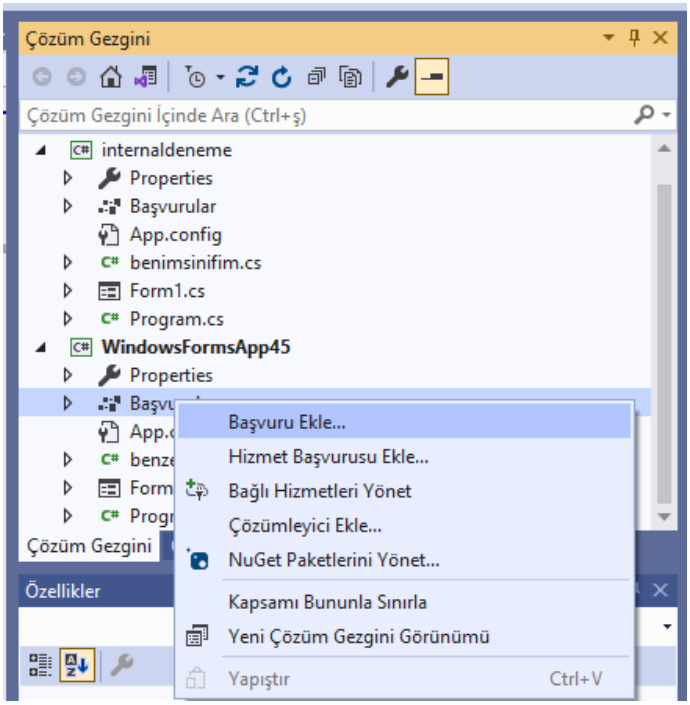


Şekil 7.11. Proje Ekleme

Farklı projelerin beraber çalışabilmesi için başvuru olarak eklenmesi gerekmektedir.

Bu pencerede Windows forms uygulamasını seçip *Sonraki* butonuna tıkladıktan sonra proje adına *internaldeneme* ismine vererek oluştur butonuna tıklayalım. Bu işlem ile projemize ikinci bir proje ekleme işlemini gerçekleştirmiş oluyoruz. İkinci projemize *benimsinifim* adlı bir sınıf ekleyelim.

Eklediğimiz internaldeneme isimli projenin ilk proje ile beraber çalışabilmesi için başvuru olarak eklenmesi gerekmektedir. Bu işlem çözüm gezgini üzerinde ilk projenin başvurular sekmesi üzerinde sağ tıklama yapılarak gerçekleştirilmektedir. Şekil 7.12.'de gösterildiği gibi açılan menüden *Başvuru Ekle* komutu verilmesi gerekmektedir.



Şekil 7.12. Buşvuru Ekle Komutu

Ekrana gelen *Başvuru Yöneticisi* penceresinde sol tarafta bulunan projeler sekmesi işaretlendikten sonra internaldeneme isimli projeye ait kutucuk onaylanıp *tamam* butonuna tıklanır. Böylece internaldeneme isimli projenin kodları ilk projemiz içinde kullanılabilecek şekilde işaret edilmiş olacaktır. Bu işlemin ardından yapmamız gereken son bir adım bulunmaktadır. İlk projemizin isim uzayının hemen üst kısmında bulunan alanda *using* anahtar kelimesini kullanarak eklediğimiz projenin adını yazmamız gerekmektedir. Bu işleme ait kod yapısı aşağıda gösterilmektedir.

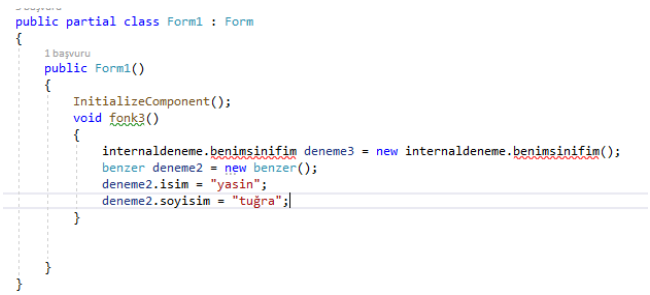
- using internaldeneme;

Eklediğimiz internaldeneme isimli projede bulunan benimsinifim isimli sınıftan, ilk proje içerisindeki *fonk3* isimli metotta deneme3 isimli bir nesne türetelim. Bu işlemi yapabilmek için ilk projemizde bulunan kodları aşağıdaki gibi düzenleyelim.

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
        void fonk3 ()
        {
            internaldeneme.benimsinifim deneme3 = new internaldeneme.benimsinifim();
            benzer deneme2 = new benzer();
            deneme2.isim = "yasin";
            deneme2.soyisim = "tuğra";
        }
    }
}
```

Erişim belirleyicisi yazılmayan veya erişim belirleyicisi internal olarak ayarlanan sınıfların farklı projelerden erişilmesine izin verilmemektedir.

Görüldüğü üzere benimsinifim isimli sınıf hata vermektedir. Bu duruma ait hata listesinde görüntülenen mesaj Şekil 7.13.'de gösterilmektedir.



Şekil 7.13. Benimsinifim Hata Çizgisi

Bunun sebebi oluşturulan sınıfların varsayılan değer olarak internal tanımlanmasından kaynaklanmaktadır. Tıpkı sınıflar içinde oluşturulan özelliklerin erişim belirleyicileri bildirilmediğinde private olarak ayarlanması gibidir. İkinci projemize eklediğimiz benimsinifim isimli sınıfın başına public erişim berlirleyicisini yazdığımız zaman bu hata mesajının ortadan kalmış olduğunu gözlemleyebilirsiniz. Erişim belirleyicisi yazılmayan veya erişim belirleyicisi internal olarak ayarlanan sınıfların farklı projelerden erişilmesine izin verilmemektedir.

Protected Internal

Protected internal erişim belirleyicisi, protected erişim belirleyicisiyle internal erişim belirleyicisinin OR (veya) operatörüyle birleştirilmesi gibi çalışmaktadır. Protected internal erişim belirleyicisiyle tanımlanan özellik veya metoda tanımlandığı sınıftan, türetildiği sınıflardan veya farklı sınıflardan tek proje içerisinde erişim mümkündür. Bu erişim belirleyicisi kullanılarak tanımlanan özelliğe farklı bir projede bulunan sınıf içinden erişilmesine izin verilmemektedir. Ancak farklı bir projede bulunan türetilmiş bir sınıf içinde erişim mümkündür.

Protected internal erişim belirleyicisi, protected erişim belirleyicisiyle internal erişim belirleyicisinin OR (veya) operatörüyle birleştirilmesi gibi çalışmaktadır.

Konunun daha iyi anlaşılması için internal erişim belirleyicisinde geliştirmiş olduğumuz uygulama üzerinden devam edelim. Projemize eklediğimiz internaldeneme isimli projedeki benimsinifim isimli sınıf içinde string veri türüne sahip protected internal olarak tanımlanan uzunluk özelliğini ayarlayalım. Kodlarımız aşağıdaki gibi olmalıdır.

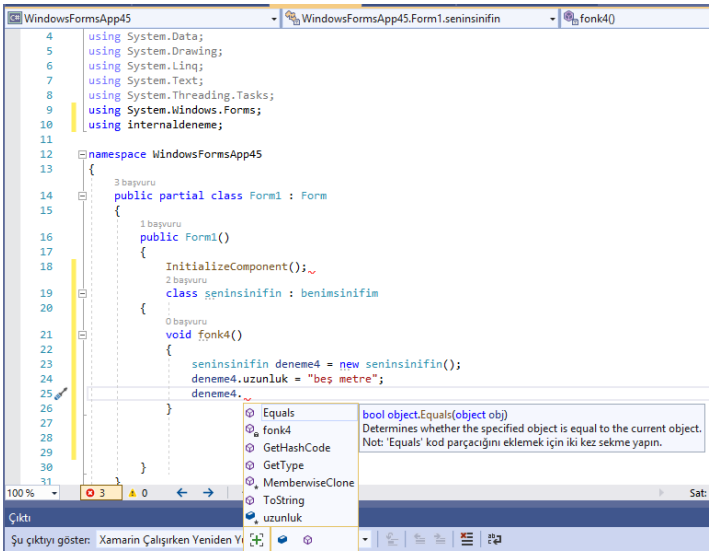
```
namespace internaldeneme
{
   public class benimsinifim
   {
      protected internal string uzunluk;
   }
}
```

Bu işlemin ardından ilk projemize gelerek *benimsinifim* isimli sınıftan *seninsinifin* isimli bir sınıf türetelim. Bu sınıf içinde *fonk4* metodunu oluşturarak *deneme4* ismiyle türettiğimiz nesnede uzunluk özelliğinin çalışıp çalışmadığını kontrol edelim. İlk projemizdeki kodları aşağıda verildiği gibi düzenleyelim.

```
public Form1()
    {
       InitializeComponent();
       class seninsinifin : benimsinifim
       {
          void fonk4()
          {
```

```
            seninsinifin deneme4 = new seninsinifin();
            deneme4.uzunluk = "beş metre";
        }
    }
}
```

Görüldüğü üzere protected internal olarak tanımlanan bir özellik veya metoda farklı sınıfta türetilen bir nesne üzerinden erişmek mümkündür. Deneme4 isimli nesnemizi yazdıktan sonra nokta koyup özellikler panelinde uzunluk seçeneğinin bulunduğunu görebiliriz. Bu durum Şekil 7.14.'de gösterilmektedir.



Şekil 7.14. Deneme4 Nesnesinin Özellikler Listesi

Protected internal olarak tanımlanan bir özellik veya metoda farklı sınıfta türetilen bir nesne üzerinden erişmek mümkündür.

Bireysel Etkinlik

- İki farklı projeyi tek bir çözüm gezgini içinde birleştirerek farklı sınıflar ve nesneler oluşturunuz. Oluşturduğunuz sınıf içinde private protected olarak tanımlayacağınız özelliğin hangi nesne ve metotlarda kullanılabildiğiniz test ederek raporlayınız.

Özet

- Bazı durumlarda nesnelerin özellikleri ve sınıflar içinde yer alan metotların kullanımı özelleştirilmek istenebilir. Yani her özelliğe erişim durumu aynı olmayabilir. Bazı özelliklerin ve metotların alt sınıflarda veya diğer projelerde kullanılıp kullanılmayacağı ayarlanabilmektedir. İşte bu işlemleri gerçekleştirmemizi sağlayan yapı erişim belirleyicileridir. Aynı zamanda erişim belirleyicileri sayesinde sınıflara erişebilme durumu da ayarlanabilmektedir.
- C# .Net içinde temel olarak kullanabileceğimiz 4 adet erişim belirleyicisi bulunmaktadır. Bu erişim belirleyicilerin birlikte kullanımı da söz konusudur. Örneğin protected ve internal erişim belirleyicilerin beraber kullanımını ifade protected internal erişim belirleyicisidir. Benzer şekilde private protected erişim belirleyicisi de private ve protected erişim belirleyicilerin birlikte kullanımı olacak şekilde ayarlanmıştır.
- Public erişim belirleyicisini kullandığımızda, kısıtlama olmayan yani sınıf içi, sınıf dışı ve farklı projeler olmak üzere her yerden erişilebilir olmasını sağlamaktayız. Tabi public olarak tanımlanan değişkenin ya da metodun farklı projelerde (exe dosyalarında) kullanılabilmesi için başvuru olarak eklenmesi gerekmektedir.
- Public olarak tanımlanan bir özelliğe kendi sınıfı içinden de ulaşılabilmektedir. Bu durumu basit bir metot tanımlayarak gösterebiliriz.
- Erişim belirleyicileri içinde en kısıtlayıcı olan erişim belirleyicisi private erişim belirleyicisidir. Sadece tanımlandığı sınıf içinden erişilmesine izin vermektedir. Bununla birlikte bir özellik veya metot yazılırken herhangi bir erişim belirleyici bildirilmediği takdirde program o özellik veya metodu varsayılan olarak Private algılamaktadır. Ancak genellikle bu belirleyicinin yazılması tavsiye edilmektedir.
- Private erişim belirleyicisi kullanılarak oluşturulan bir özelliğe farklı sınıflardan erişilemeyeceğini söyledik. Public erişim belirleyicisinde olduğu gibi farklı bir sınıf içinde yazmak istersek çalıştırma aşamasına geçmeden kod editöründe kırmızı bir alt çizgi ile program bizi uyarmaktadır.
- Protected erişim belirleyicisi kullanılarak tanımlanan özellik ve metotlara aynı sınıf içinden erişilebilmektedir. Ancak farklı sınıflardan bu özellik ve metotlara erişim engellenmektedir. Sadece bu yönüyle Protected erişim belirleyicisini değerlendirecek olursak Private erişim belirleyicisinden bir farkı olmadığı düşünülebilir. Bu durumda kalıtım kavramı devreye girmektedir. Yani Protected erişim belirleyicisinin private erişim belirleyicisinden farkını anlayabilmek için kalıtım kavramını kullanacağımız bir uygulama geliştirmemiz gerekmektedir.
- Protected olarak tanımlanan özellik ve metotlara o sınıftan türetilmiş sınıflardan da erişebilmek mümkündür. Private erişim belirleyicisinde böyle bir durum söz konusu değildir.
- Tek bir proje üzerinde çalışırken internal erişim belirleyicisi ile public erişim belirleyicisi arasında bir fark bulunmamaktadır. Internal erişim belirleyicisi kullanılarak tanımlanan özellik ve metotlara aynı sınıftan, türetilen sınıflardan ve farklı sınıflardan erişim mümkündür. Buradaki fark iki projenin tek bir çözüm gezgini üzerinde çalışırken ortaya çıkmaktadır.
- Protected internal erişim belirleyicisi, protected erişim belirleyicisiyle internal erişim belirleyicisinin OR (veya) operatörüyle birleştirilmesi gibi çalışmaktadır. Protected internal erişim belirleyicisiyle tanımlanan özellik veya metoda tanımlandığı sınıftan, türetildiği sınıflardan veya farklı sınıflardan tek proje içerisinde erişim mümkündür. Bu erişim belirleyicisi kullanılarak tanımlanan özelliğe farklı bir projede bulunan sınıf içinden erişilmesine izin verilmemektedir. Ancak farklı bir projede bulunan türetilmiş bir sınıf içinde erişim mümkündür.

DEĞERLENDİRME SORULARI

1. Aşağıdakilerden hangisi C# .Net içinde temel olarak kullanabileceğimiz erişim belirleyicilerinden biri değildir?
   a) Public
   b) Private
   c) Protected
   d) Internal
   e) Using

   Sınıflar ……. ya da …….. olarak nitelendirilemezler.
2. Cümlede boş bırakılan yerlere aşağıdakilerden hangisi getirilmelidir?
   a) Public – Protected
   b) Protected – Private
   c) Protected - Internal
   d) Protected – Protected Internal
   e) Public – Internal

3. Aşağıdaki erişim belirleyicilerinden hangisini kullanırsak tanımlanan özellik veya metoda her yerden erişim sağlayabiliriz?
   a) Public
   b) Private
   c) Protected
   d) Internal
   e) Protected Internal

4. Tasarım alanında çalışırken kod editörüne geçmek için hangi fonksiyon tuşunu kullanmamız gerekir?
   a) F3
   b) F4
   c) F5
   d) F6
   e) F7

5. Kod editöründe türetilen nesnenin özelliklerini görmek için hangi operatörü kullanmamız gerekmektedir?
   a) İki nokta üst üste (:)
   b) Noktalı virgül (;)
   c) Nokta (.)
   d) Ünlem (!)
   e) Soru işareti (?)

6. Aşağıdakilerden hangisi erişim belirleyicileri içinde en kısıtlayıcı olan erişim belirleyicisidir?
   a) Public
   b) Private
   c) Protected
   d) Internal
   e) Protected Internal

7. Bir sınıftan kalıtım yoluyla başka bir sınıf türetebilmek için aşağıdaki operatörlerden hangisini kullanmamız gerekir?
   a) Noktalı virgül(;)
   b) İki nokta üst üste (:)
   c) Virgül (,)
   d) Nokta (.)
   e) Çift tırnak (“”)

8. Eklediğimiz projeleri kod editöründe tanımlamak için hangi anahtar kelimeyi kullanmamız gerekmektedir?
   a) Class
   b) Private
   c) Namespace
   d) Using
   e) Public

   Protected internal erişim belirleyicisi, protected erişim belirleyicisiyle internal erişim belirleyicisinin …….. operatörüyle birleştirilmesi gibi çalışmaktadır.

9. Cümlede boş bırakılan yere aşağıdakilerden hangisi getirilmelidir?
   a) And (ve)
   b) Not (değil)
   c) Eşittir (=)
   d) Or (veya)
   e) Nokta (.)

10. Protected internal olarak tanımlanan bir özellik veya metoda hangi sınıftan ulaşılamaz?
    a) Aynı proje içinden – tanımlandığı sınıftan
    b) Aynı proje içinden – türetilmiş sınıftan
    c) Aynı proje içinden – diğer sınıflardan
    d) Farklı proje içinden – diğer sınıflardan
    e) Farklı proje içinden – türetilmiş sınıftan

Cevap Anahtarı

1.e, 2.b, 3.a, 4.e, 5.c, 6.b, 7.b, 8.d, 9.d, 10.d

YARARLANILAN KAYNAKLAR

Aktaş, V. (2010). Visual studio 2010 ile her yönüyle C# 4.0 (2. Baskı). İstanbul: Kodlab Yayın.

Aktaş, V. (2020). Visual studio 2019 (1.Baskı). İstanbul: 01 Yayınları.

Atasever, V. (2017). C# 7 Uygulamalarla C# programlama dilini keşfedin (2. Baskı). Kocaeli: Level Kitap.

Demirli, N. & İnan, Y. (2005). Visual C# .Net (3. Baskı). Ankara: Palme Yayıncılık.

Purdum, J. (2013). Beginning Object-Oriented Programming with C#. Indianapolis: John Wiley & Sons, Inc.

Sharp, J. (2011). Adım adım Microsoft visual C# 2010. (Çev. T. Buldu). Ankara: Arkadaş Yayınevi.

Uzunköprü, S. (2017). Projeler İle C# 7.0 ve SQL Server 2016 (7. Baskı). İstanbul: Kodlab Yayın.

Vatansever, F. (2004). Algoritma Geliştirme ve Programlamaya Giriş (2. Baskı). Ankara: Seçkin Yayıncılık.

Yanık, M. (2008). Visual studio 2008 ile Microsoft visual C# 3.0 for .Net framework 3.5 (1. Baskı). Ankara: Seçkin Yayıncılık.

Yücedağ, M. (2020). C# Eğitim kitabı (3. Baskı). İstanbul: Dikeyeksen.