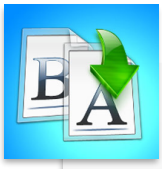
# METOTLARA GİRİŞ



İÇİNDEKİLER

- Metot Nedir?
- Metotların Kullanımı
  - Metotlarda Özel İfade ve Yapılar
- Metotlarda Erişim Belirleme

HEDEFLER

- Bu üniteyi çalıştıktan sonra;
  - Metot kavramını ve önemini kavrayabilecek,
  - Sınıflarda metot kavramının kullanışı ve faydalarını kavrayabilecek,
  - Metotlarda erişim seviyesine göre işlem yapma yöntemlerini anlayabilecek,
  - Algoritma mantığında metotların kullanılmasının faydalarını anlayabilecek,
  - Public, protected ve private kavramlarının güvenlik anlamında kullanım yöntemlerini anlayabileceksiniz.

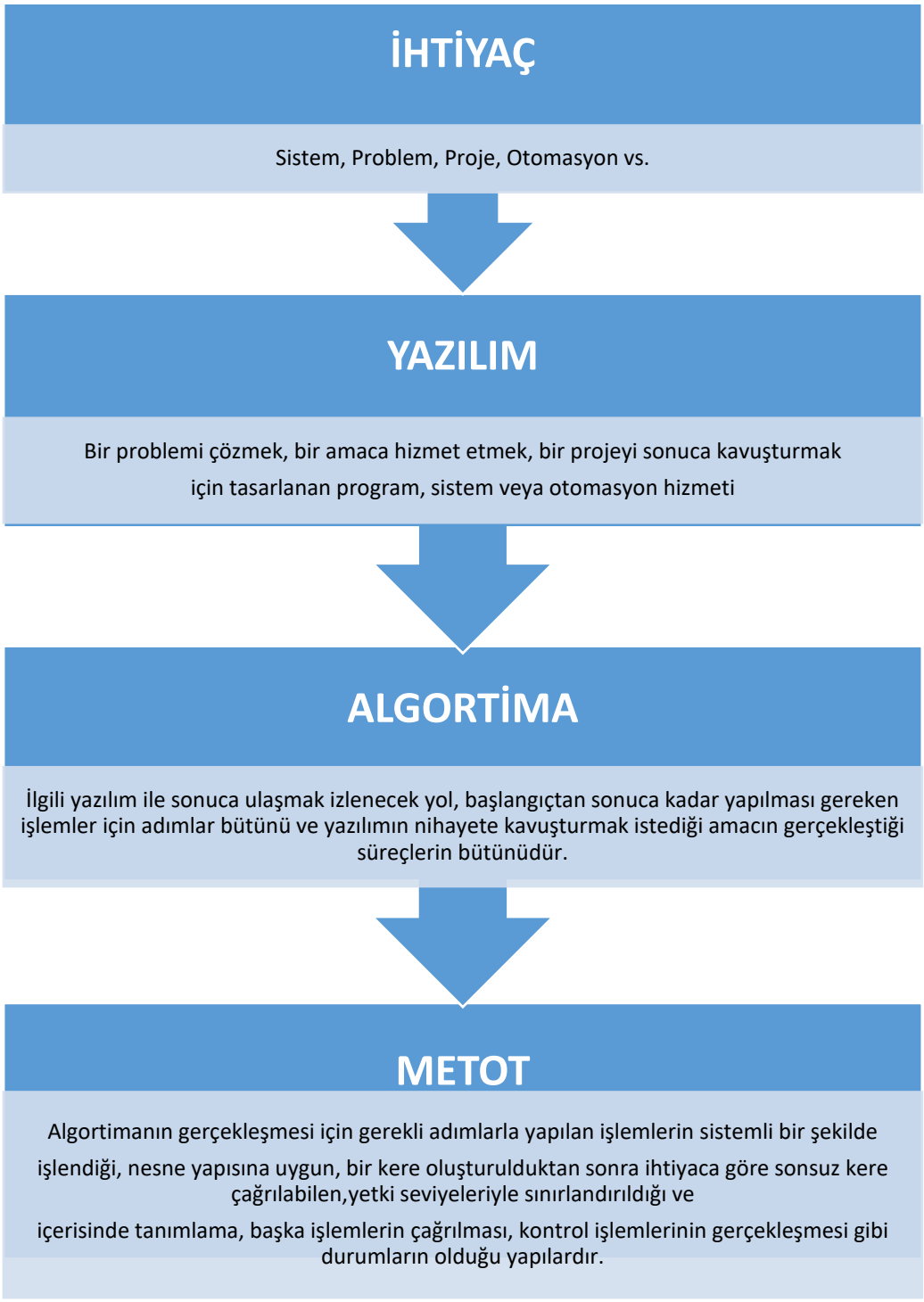
Atatürk Üniversitesi
Açıköğretim Fakültesi

NESNE TABANLI PROGRAMLAMA I

Dr. Öğr. Üyesi
Muhammed Fatih ALAEDDİNOĞLU

ÜNİTE

8

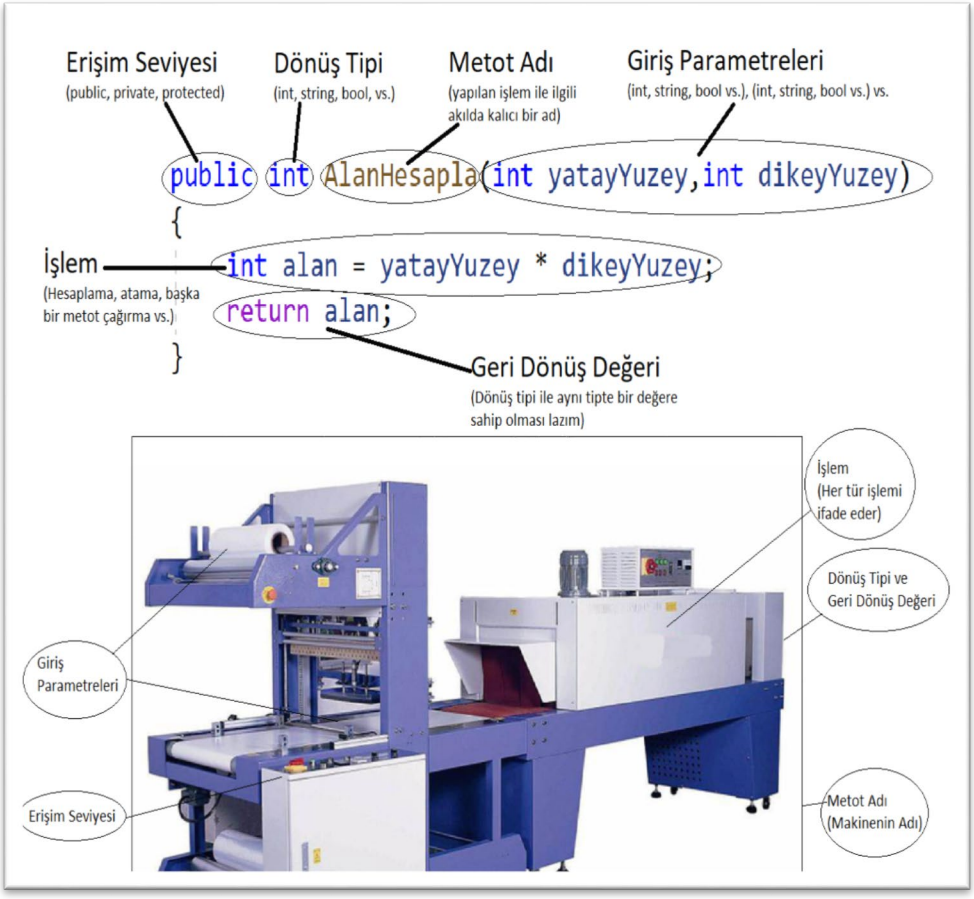

İHTİYAÇ
Sistem, Problem, Proje, Otomasyon vs.
YAZILIM
Bir problemi çözmek, bir amaca hizmet etmek, bir projeyi sonuca kavuşturmak
için tasarlanan program, sistem veya otomasyon hizmeti
ALGORTİMA
İlgili yazılım ile sonuca ulaşmak izlenecek yol, başlangıçtan sonuca kadar yapılması gereken işlemler için adımlar bütünü ve yazılımın nihayete kavuşturmak istediği amacın gerçekleştiği süreçlerin bütünüdür.
METOT
Algortimanın gerçekleşmesi için gerekli adımlarla yapılan işlemlerin sistemli bir şekilde
işlendiği, nesne yapısına uygun, bir kere oluşturulduktan sonra ihtiyaca göre sonsuz kere çağrılabilen,yetki seviyeleriyle sınırlandırıldığı ve
içerisinde tanımlama, başka işlemlerin çağrılması, kontrol işlemlerinin gerçekleşmesi gibi durumların olduğu yapılardır.

# GİRİŞ

Metotlar yapısı gereği bir makine, bir alet veya bir fabrikaya benzemektedir.

Metotlar, nesne tabanlı programlamanın en önemli öğelerinden biri olmakla beraber, kullanım mantığı ve gerçekleştirdiği işlemler açısından çok farklı özellikleriyle hemen hemen her üst seviye dilin en etkin yapılarındandır. *Metotlar kelime anlamı itibariyle yöntem, sistem, teknik, prosedür ve işlem gibi anlamlara gelmektedir.* Nesne tabanlı programlama kapsamında metotların temel kullanım amacı, daha anlaşılır, fonksiyonel, kullanışlı ve basit çalışan kodlar üretmektir.

*Metotlar yapısı gereği bir makine, bir alet veya bir fabrikaya benzetilebilir.* Bir makine, alet veya fabrikaya hammadde olarak içeriden veya dışarıdan verilen değer veya değerler, belirli işlemlerden geçtikten sonra bir çıktı üretmektedir. Bu durum Şekil 8.1'de metotlar ile sanayide kullanılan bir cihazın karşılaştırması şekilde özetlenmiştir.

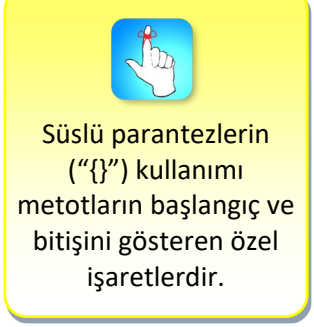



**Şekil 8.1.** Metotları Anlatan Örnek Resim

Günümüzde hemen hemen bütün işletme ve kuruluşlar, işlemlerini belirli prosedür ve adımlar ile yapmaktadır. Yapılan bu işlemlerin çoğunluğu ise standart yapılarda yani aynı girdiye aynı çıktıyı verecek şekillerde gerçekleşmektedirler. Örneğin; bir dikdörtgenin alanını hesaplarken dik ve yatay uzunlukları çarparız. Bu işlemin sonucu ve alanın hesaplanması aynı girdi verileriyle hep aynı sonucu verecektir. Böyle bir senaryoyu program haline getirdiğimizde ise metotlara başvururuz. *Program kapsamında yapılması gereken işlemler, metotlar haline getirildiği takdirde hem yapılan işlemin anlaşılması kolaylaşır hem aynı kodlar her*

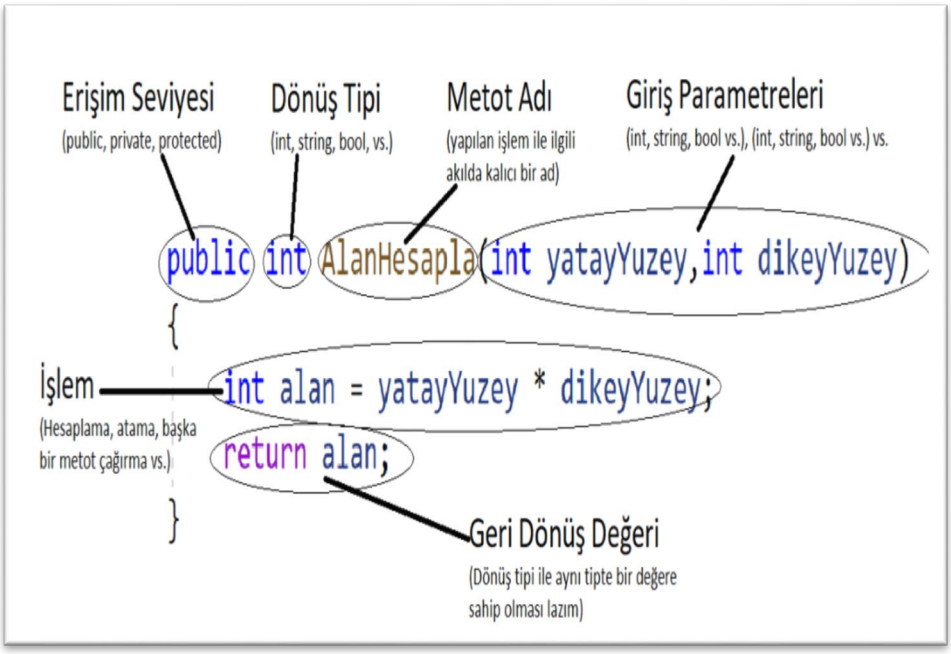
*defasında yazmak zorunda kalınmaz.* Ayrıca başarılı program ve projeler için metotların kullanılması, anlaşılması ve geliştirilmesi çok önemli bir durumu ifade etmektedir. Genellikle orta ya da büyük ölçekte bir yazılım zaman içerisinde doğabilecek ihtiyaçlara cevap verebilmek adına birden fazla kişinin müdahalesine ihtiyaç duyar. Böyle bir durumda kod ile ilgilenen kişi, metodu ilk yazan kişiden farklı birisi olsa bile, ilgili metoda hızlı ve kolay bir şekilde müdahale edebilir.

## METOT NEDİR?

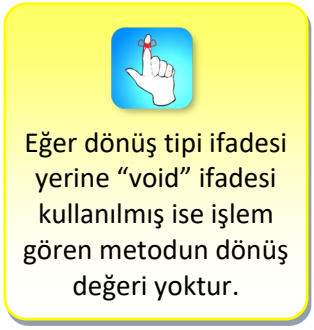


Süslü parantezlerin (“{}”) kullanımı metotların başlangıç ve bitişini gösteren özel işaretlerdir.

*Yöntem, sistem, teknik, prosedür ve işlem gibi anlamlara gelen metot, gerçekleştirilen programda, yazılımda veya projede sınıf içerisinde belirli bir işi veya işleri yerine getirmek için kullanılan kod parçalarıdır.* Bu kod parçaları belirli yapılarda tanımlanır. Bu yapıları genel olarak Şekil 8.2.’deki gibi tanımlarız.




**Şekil 8.2.** Metotlara Ait Genel Bir Yapı



Eğer dönüş tipi ifadesi yerine “void” ifadesi kullanılmış ise işlem gören metodun dönüş değeri yoktur.

İlk olarak metotlar oluşturulurken öncelikle erişim seviyesi tanımlanır. Daha sonra dönüş tipi ve metot adı tanımlandıktan sonra normal parantezler içerisine eğer gerekli ise metot içerisinde kullanılacak parametreler belirtilir. Birden fazla giriş parametresi kullanılması gerekiyorsa aralarına virgül koyulur. Ayrıca giriş parametreleri, değişken tipi ve adı olmak üzere ikili yapıdan oluşmaktadır. Bu adımlardan sonra işlem görecek kodlar süslü parantezler (“{}”) içerisine yazılır. *Son olarak da yapılacak bütün işlemlerden sonra eğer dönüş tipi belirtilmiş ise “return” ifadesi ile birlikte ilgili veri veya bilgi aynı tipte olacak şekilde yazılır.* Bu kısımda işlem satırının bitimini ifade eden “;” işareti koymak suretiyle metot tamamlanır.

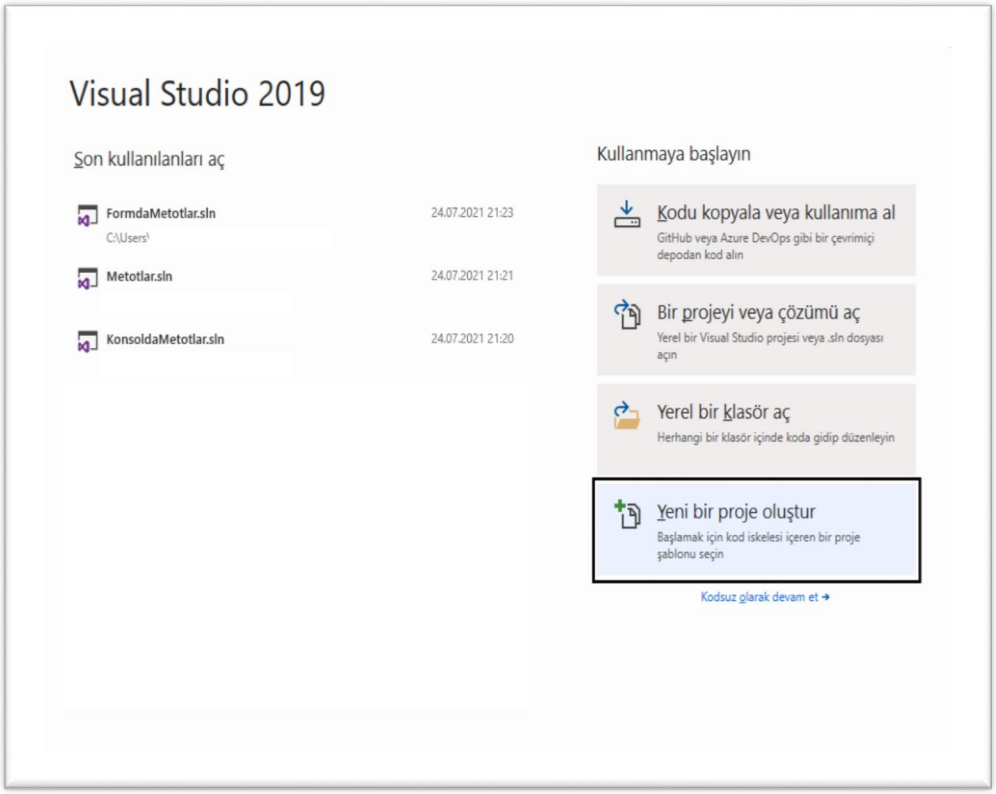
Metotların ilk oluşturulurken öncelikle erişim seviyesi tanımlanır.

Süslü parantezlerin (“{}”) kullanımı “C#” programlama dilinde metotların başlangıç ve bitişini gösteren özel işaretlerdir. *Ayrıca dönüş tipi ifadesi yerine “void” ifadesi kullanılmış ise işlem gören metodun dönüş değeri yoktur. Sadece çalıştırılabilir bir yapıya sahiptir.* Metotlarda dönüş tipi olarak string, double, int, char, bool, dizi ifadeleri(string[] gibi)ve dönüşün olmadığını ifade eden void anahtar kelimesi kullanılmaktadır. Bu tiplerden string metinsel, bool mantıksal ifadeler için kullanılırken int, double ve float ise farklı özelliklerdeki sayısal değerler için kullanılır.

Nesne tabanlı programlama mantığında sınıflar içinde tanımlanan metotlar, sınıfın amacına bağlı olarak oluşturulmuş işlemlerdir. Sınıf ve metotlardaki erişim seviyeleri, yapısal esneklikler ve anlamlı isimlendirmeler güvenli ve düzenli kod yazılımına olanak sağlamaktadır. Bu bölümde metotların daha iyi anlaşılabilmesi için örnek uygulamalar, Visual Studio platformu C# programlama dilinde yapılacaktır. Yapılacak örnekler form ekranında gerçekleştirilerek konunun daha iyi anlaşılması sağlanacaktır.

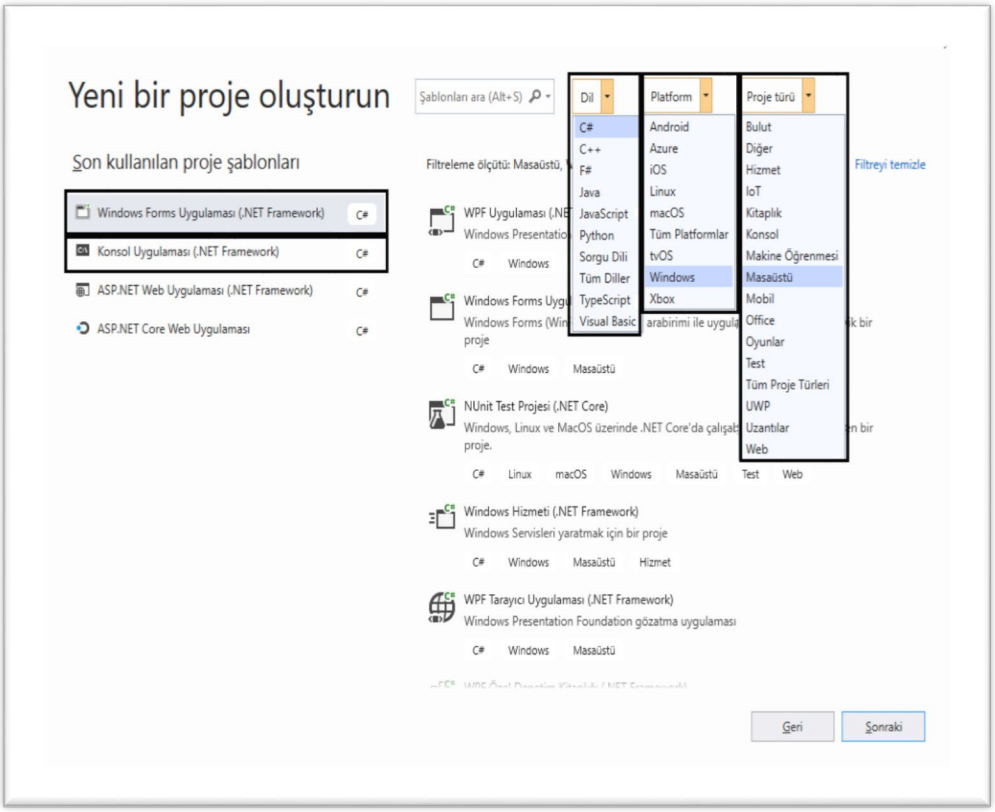
## METOTLARIN KULLANIMI

Ünite kapsamındaki örnekler Visual Studio platformu üzerinde C# programlama dili kullanılarak verilecektir. Bunun için ilk olarak Visual Studio platformunda proje açma ile başlayalım.




**Resim 8.1.** Visual Studio'da Yeni Proje Açma

Resim 8.1.'de "Yeni bir proje oluştur" seçeneği işaretlenerek yeni proje için ilk adım gerçekleşmiş olacaktır. Daha sonra Resim 8.1.'de görülen adımlar takip edilebilir.

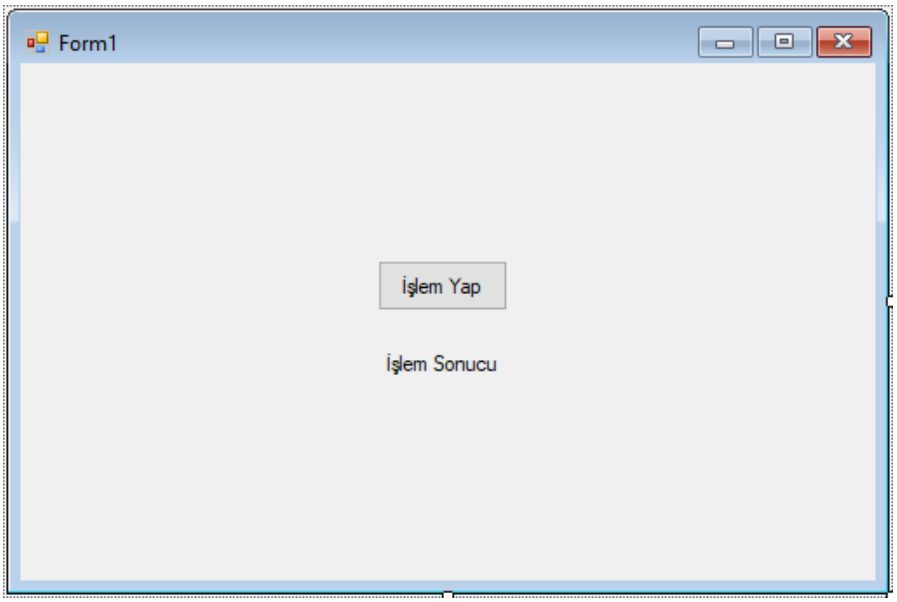



**Resim 8.2.** Visual Studio'da Proje Şablonu, Dil, Platform Ve Proje Türü Seçimi

Resim 8.2.'de görüldüğü gibi ilk olarak proje şablonu belirlenir. Ardından projede kullanılacak programlama dili "C#" olarak seçilir. Daha sonra projenin gerçekleşeceği platform bilgisi olarak "Windows" seçeneği işaretlenir. Son olarak da projenin türü yani uygulamalarımızı gerçekleştireceğimiz Masaüstü bilgisi seçilir. Bu kısımda önemli bir bilgi olarak daha önce de belirttiğimiz gibi proje şablonu kısmında "Windows Form Uygulaması" seçilecek şekilde proje oluşturulmuştur. Ancak bu ünite kapsamında verilen örnekler isteğe bağlı olarak "console" uygulamalarında da denenebilir. Sonraki butonuna bastıktan sonra "Program.cs" adında bir sınıf ve "Form1" sayfası otomatik oluşturulur. Bu "Program.cs" sınıfı proje çalıştığı zaman ilk olarak derlenen ve projenin başlayacağı nokta olarak işlem görür.

Program.cs proje çalıştığı zaman ilk olarak derlenen ve projenin başlayacağı nokta olarak işlem görür.

Ünite kapsamında yapacağımız örneklerimizin daha iyi anlaşılması adına oluşturulan projede Resim 8.3.'de gösterildiği gibi "Form1" ekranında işlemleri tetikleyen buton(İşlem Yap) ve sonuçları gösterebileceğimiz text (label-İşlem Sonucu) alanları eklenmiştir.



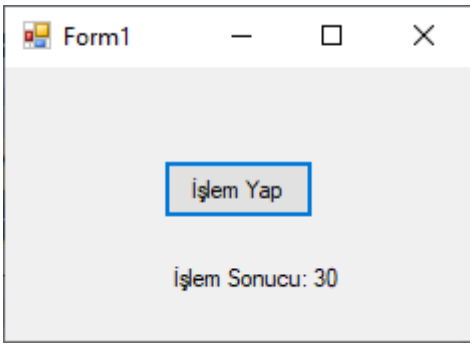

**Resim 8.3.** Form1 Ekranı

Örnek

•Form1 ekranında butona tıklandığı zaman AlanHesapla metodunun çağrılması ve ilgili işlem sonucunun çıktısı gösterilecektir.

```
public int AlanHesapla(int yatayYuzey, int dikeyYuzey)
{
   int alan = yatayYuzey * dikeyYuzey;
   return alan;
}
private void IslemYap_Click(object sender, EventArgs e)
{
     IslemSonucu.Text= "İşlem Sonucu: " + AlanHesapla(5,6).ToString();
}
```

Yukarıdaki kodlarda gösterildiği gibi ilk olarak “AlanHesapla” adında bir metot tanımlıyoruz. Daha sonra “Form1” ekranındaki “İşlem Yap” butonuna tıklandığı zaman “AlanHesapla” metodu içerisindeki işlem gerçekleştikten sonra sonuç değerini “IslemSonucu” alanına Resim 8.4.’de gösterildiği gibi yazdırıyoruz.



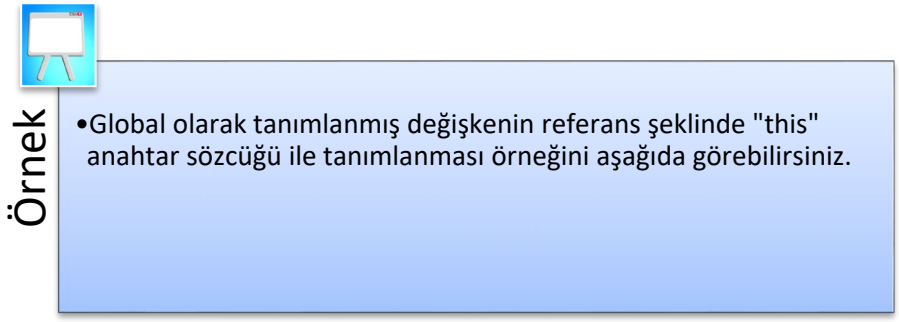

**Resim 8.4.** Alanhesapla Metoduna Ait Dönen Sonuç

Yukarıdaki örnekte metotların kullanımına ait işlemlerin sonucu Resim 8.4.'de gösterilmiştir.

## Metotlarda Özel İfade ve Yapılar

### This anahtar sözcüğü

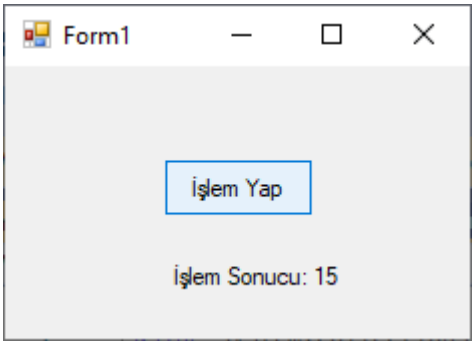
*This anahtar sözcüğü global (genel) olarak tanımlanmış değişkenlerin referans ile metot içerisinde kullanılmasını sağlar.* Yani sınıf içerisinde tanımlanan değişkenin sınıfa ait olduğunu ifade eden bir referans yapısı olarak tanımlanır. Böylece metot içerisinde benzer isimde tanımlanan değişkenler ayrılır.




```
public partial class Form1 : Form
{
    int sonuc = 0;

    public int SonucEkle(int sonuc)
    {
      this.sonuc = this.sonuc + sonuc;
      return this.sonuc;
    }

    private void IslemYap_Click(object sender, EventArgs e)
    {
      IslemSonucu.Text="İşlem Sonucu: "+ SonucEkle(15).ToString();
    }
}
```

Örnekte gösterildiği gibi sınıf içerisinde global olarak tanımlanan "sonuc" değişkeni "SonucEkle" metodu içerisinde "this" anahtar kelimesi ile birlikte kullanılmıştır. Aynı zamanda "SonucEkle" metodunun giriş parametresi de "sonuc"

kelimesi olarak ifade edilmiştir. Böyle durumlarda oluşturulan değişken ismi program için genel olarak mı yoksa metodun giriş parametresi olarak mı ifade edilmiş karışacağı için "this" anahtar sözcüğü kullanılır. Bu sözcük genel yani global olarak tanımlanmış değişkeni kullanabilmek için eklenir. Resim 8.5.'de gösterildiği gibi butona tıklandığı zaman "SonucEkle" metodu için verilen yerel giriş parametresi (15 değeri), global olarak ifade edilen "sonuc" değerine atanır. Daha sonra bu değer işlem sonucuna atanır.




**Resim 8.5.** "This" Anahtar Sözcüğünün Kullanım Şekli

## Main metodu

Bu metot projenin yani C# uygulamasının başlangıç noktasıdır. Yani geliştirilen program derlenmeye başladığı anda Main metodunu bularak bu kısımdan başlar. Program içerisinde yalnızca bir tane *Main metodu vardır. Main metoduna ait aşırı yüklemeler (overloading) yapılabilir. Yani Main metodu argüman (giriş parametreleri) verilmeden, erişim seviyesi değiştirilerek (standartta "private" iken "public" veya "protected" olabilir) işlem yapılabilir.* Ayrıca geri dönüş değeri verilecek şekilde güncellenebilir. Ancak bu metot kesinlikle "static" şekilde tanımlanmalıdır.

Main metodu projenin yani C# uygulamasının başlangıç noktasıdır.

## Program sınıfı

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

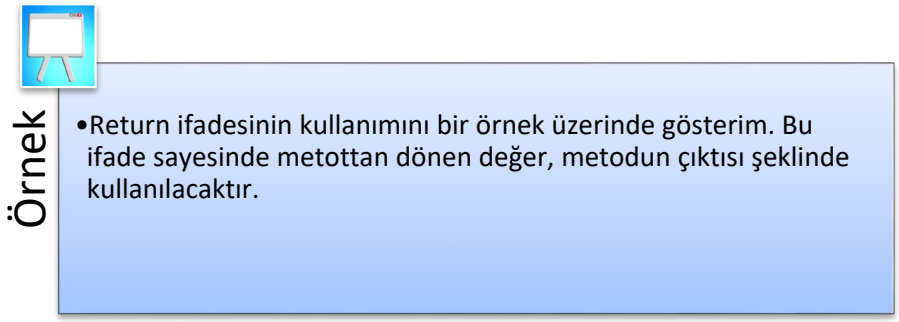
Void ifadesi oluşturulan metodun geri değer döndürmeyeceği anlamını taşır.

## Void ifadesi

Bu ifade oluşturulan metodun geri değer döndürmeyeceği anlamını taşır. Yani oluşturulan metot bir takım işlemler yaptıktan sonra çıkan sonuçlardan herhangi birinin metodun bir çıktısı olarak ifade edilmemesi olayıdır. *Ünitenin başında belirttiğimiz gibi metodu bir fabrika içindeki makinelere benzetirsek, üretilen mamuller fabrika içerisinde başka birimlerin veya makinelerin kullanımına sunulduktan sonra çıkış ünitesinden herhangi bir ürün çıkmamaktadır. Benzer şekilde yapılan programda da oluşturulan metodun program içerisinde bir takım değerleri oluşturup veya kontrolleri sağladıktan sonra çıkış değeri olarak herhangi bir sonuç geri döndürmemesi "void" ifadesi gibi anlaşılabilir.*

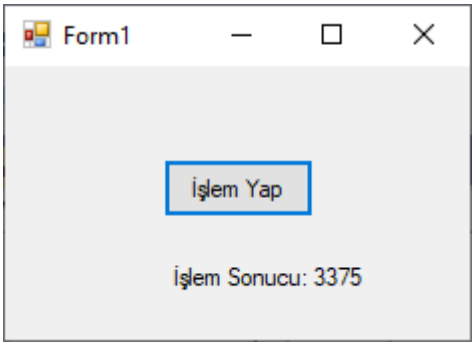
## Return ifadesi

*Bu ifade nesne tabanlı programlamada sıkça kullanılan ve yapılan işlemlerden sonra istenilen değerin geri gönderilmesi anlamında kullanılmaktadır.* Özellikle metotlarda kullanılan bu ifade metot içerisinde bir takım işlemlerden sonra istenilen değerin -metodun veri tipi ile de uyumlu olacak şekilde- metodun sonucu olarak çıktı değer vermesi şeklinde kullanılabilir. Bu ifadeyi bir örnek ile açıklamak uygun olacaktır.




```
public int ReturnSonuc(int sonuc)
{
  return sonuc*sonuc*sonuc;
}

private void IslemYap_Click(object sender, EventArgs e)
 {
  IslemSonucu.Text="İşlem Sonucu: "+ ReturnSonuc(15).ToString();
 }
```

Örnekte gösterildiği gibi "ReturnSonuc" metodundan dönen sonuç değeri "IslemSonucu" alanına atanmıştır. Daha sonra bu değişken Resim 8.6.'daki gibi form ekranına değer olarak yazdırılmıştır.




**Resim 8.6.** Return İfadesinin Kullanımına Ait Çıktı

## Aşırı yüklemeler(Overloading)

Aşırı yüklemeler C# dilinde çok yaygın ve etkin kullanıma sahiptir.

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

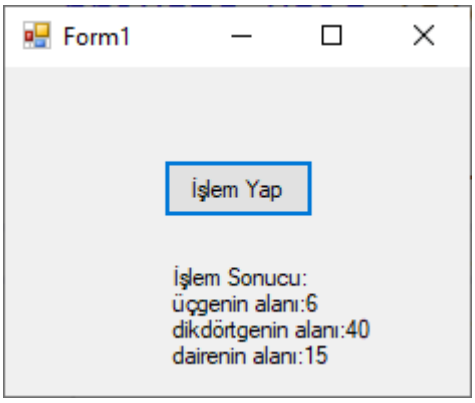
Örnek

•Aşırı yüklemeler(overloading) konusunun anlaşılması adına örnek bir uygulamayı göstermek uygun olacaktır. Bu örnekte bir metodun aynı isimde farklı görevlerde kullanılması gösterilmektedir.

```
public double AlanHesapla (int kenar1, int kenar2, double aci)
{
  double b = (aci * (Math.PI)) / 180;
  return ((double)kenar1 * (double)kenar2 * Math.Sin(b)) / 2;
 }

public int AlanHesapla (int yatayKenar, int dikeyKenar)
{
  return yatayKenar * dikeyKenar;
}

public int AlanHesapla (int yariCap)
{
 return (int)(Math.PI * yariCap);
}

private void IslemYap_Click(object sender, EventArgs e)
{
  IslemSonucu.Text="İşlem Sonucu: "+"\n" +
    "üçgenin alanı:" +AlanHesapla (4, 6, 30.0)+"\n"
     +"dikdörtgenin alanı:"+ AlanHesapla (5, 8) + "\n"
     +"dairenin alanı:"+AlanHesapla (5);
}
```

Örnekte gösterildiği gibi “AlanHesapla” metodu üç defa tanımlanmıştır. Ancak her tanımlamada giriş parametreleri farklı tip veya sayıda olduğundan aynı isimde tanımlanmaktadır. Bu durum C# programlama dilinde Overloading olarak

tanımlanmakta ve hatasız bir şekilde çalışmaktadır. Resim 8.7.'de gösterildiği gibi bir sonuç sayfasına ulaşılmaktadır.




**Resim 8.7.** Aşırı Yükeleme (Overloading) İle Aynı İsme Sahip Farklı Metotlara Ait Sonuçlar

**Static ifadesi**

C# dilinde sıkça karşımıza çıkan bu ifade genellikle tanımlı olduğu sınıfa ait bir değeri veya sonucu vermesi için kullanılır. Yani "static" olarak oluşturulan bir metot veya bir değer sınıfa ait nesne tanımlanmaksızın direk olarak sınıfın altında çağrılarak kullanılır. Bir metot "static" olarak tanımlandıktan sonra erişim seviyesine bağlı olarak kendi sınıfının içinde direk çağrılır. Ancak farklı bir sınıf içerisinde erişim seviyesine bağlı olarak (Erişim seviyesi "public" ise bütün sınıflarda, "protected" ise kendi sınıfının altında oluşturulan sınıflarda, "private" ise sadece kendi sınıfında kullanılır.) kullanılır.

# METOTLARA ERİŞİM

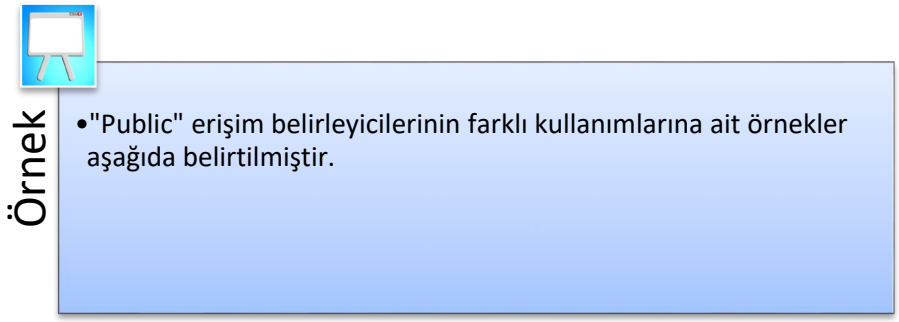
"Public, Protected ve Private" erişim belirleyicileri metotlar için çok önemlidir.

C# dili ile geliştirilmiş bir programda bir metoda erişip onun kullanımı belirli şartlara bağlıdır. Bu şartlar program ilk geliştirildiği andan itibaren C# diline özel erişim belirleyicileri sayesinde gerçekleştirilir. Bu erişim belirleyicileri aslında bir çeşit erişim seviyesi olarak da ifade edilebilir. Dolayısıyla metot tanımlamak için kullanılan erişim belirleyiciler, tıpkı değişken ya da sınıf tanımlamak için kullanılanlar gibidir. Standart olarak tanımlanmış ve en sık kullanılan "Public, Protected ve Private" ifadeleri en genel erişim izninden özel izinlere kadar olan yapıyı kapsar. Her ne kadar "Public, Protected ve Private" ifadeleri çok yaygın kullanılsa da "Internal ve Protected Internal" erişim belirleyicileri de amaca göre kullanılmaktadır.

## Public

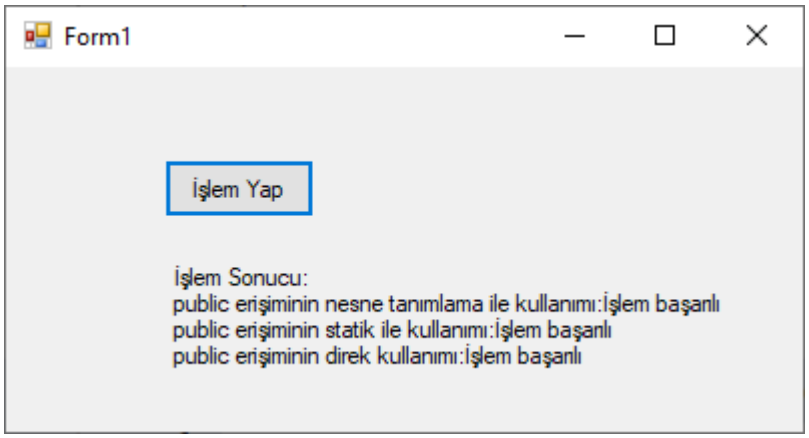
"Public" ifadesi bir metot ile ilgili erişilebilirlik açısından en esnek ve genel yapıyı ifade eder. *Yani bir sınıf içerisinde "public" türünde bir metot tanımlanmışsa o metoda hemen hemen her şartta erişim sağlanabilir demektir.* Genellikle genel işlemlerde yani metoda erişilmesi durumunda problem olmayacak, yapısal değişikliğe sebep olmayacak görevlerin gerçekleştiği durumlarda kullanılmaktadır. Aksi halde aynı projede görev alan fakat sınıflar üzerinde işlem yapan kişiler, bir şekilde programın doğru çalışmasına zarar verebilirler. Böyle durumların önüne

geçmek için “public” erişim belirleyicilerinin kullanımına dikkat etmek gerekmektedir. Bu ifade ile tanımlı bir metoda aynı sınıf içerisinden direk erişilebildiği gibi nesne tanımlayarak da erişim sağlanabilir. Aynı zamanda “public” olarak tanımlanan bu metot “static” şeklinde ifade edildiği durumlarda nesne tanımlamadan direk sınıf üzerinden de erişim mümkün olmaktadır. Benzer şekilde tanımlanan “public” metoda başka sınıftan erişim sağlamak için nesne tanımlama yoluyla veya “static” olmasına bağlı olarak sınıf üzerinden direk erişim sağlanabilir.




```
public string publicDeneme1()
{
   return "İşlem başarılı";
}
public static string publicDeneme2()
{
   return "İşlem başarılı";
}
public static string publicDeneme3()
{
   return "İşlem başarılı";
}

private void IslemYap_Click(object sender, EventArgs e)
{
 Form1 form1 = new Form1();
 IslemSonucu.Text="İşlem Sonucu: "+"\n"
  + "public erişiminin nesne tanımlama ile kullanımı:" +form1.publicDeneme1() + "\n"
  + "public erişiminin statik ile kullanımı:" + publicDeneme2() + "\n"
   + "public erişiminin direk kullanımı:" + Form1.publicDeneme3();
}
```

Örnekte “public” erişim şekline ait aynı sınıf içerisinde değişik kullanımlarına ait sonuçları Resim 8.8.’de gösterilmiştir.




**Resim 8.8.** "Public" Erişim Belirleyicilerinin Çeşitli Kullanımlarına Ait Sonuçlar

Yukarıdaki örnekte ifade edildiği gibi "public" erişim belirleyicileri ile ilgili metoda aynı sınıf içerisinde iken her türlü erişim sağlanmaktadır. Metotlar aynı sınıf (Form1 sınıfı) içerisinde yer aldığından nesne tanımlayarak, metodun ismini direk yazarak ve "static" ifadesini kullandıktan sonra "Form1" sınıfının altında çağırarak kullanmak mümkündür.

## Protected

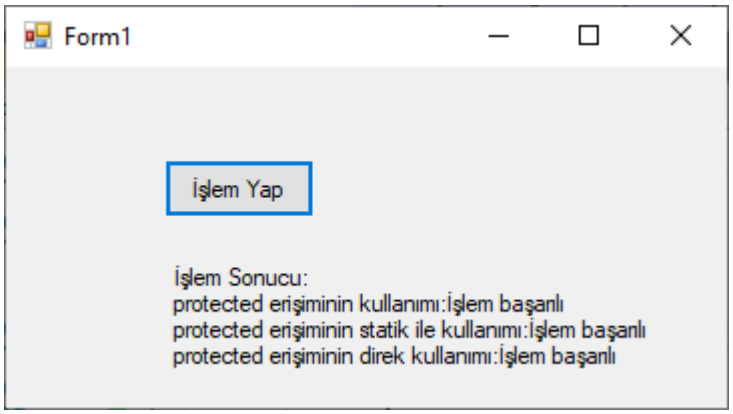
Protected erişim belirleyicisi ile oluşturulmuş metotlar sınırlı kullanıma sahiptir.

Protected ifadesi ile oluşturulan metoda erişim kısmen engellenmiş olur. *Oluşturulan başka bir sınıf, "protected" olarak ifade edilen metodun bulunduğu sınıftan türetildiği takdirde "public" ile aynı mantıkta sonuçlara ulaşmak mümkün olmaktadır.* Yani kendi sınıfı içerisinde "public" ile aynı sonuçlara erişildiği gibi farklı sınıflarda benzer sonuçlara ulaşmak mümkündür. "Protected" erişim belirleyicilerini daha iyi anlamak için farklı sınıflardaki kullanımına ait örnek yapmak uygun olacaktır.

Örnek

- "Protected" erişim belirleyicilerinin farklı kullanımlarına ait örnekler aşağıda belirtilmiştir.

```
class Erisim:Form1
{
  public string MetotErisim()
   {
    return "İşlem Sonucu: "+"\n"
     + "protected erişiminin kullanımı:" + protectedDeneme1() + "\n"
     + "protected erişiminin statik ile kullanımı:" +        protectedDeneme1() + "\n"
     + "protected erişiminin direk kullanımı:" + Form1.protectedDeneme3();
    }
}

public partial class Form1 : Form
{
  public Form1()
   {
    InitializeComponent();
    }
private void IslemYap_Click(object sender, EventArgs e)
{
   Erisim erisim = new Erisim();
    IslemSonucu.Text = erisim.MetotErisim();
}
  protected string protectedDeneme1()
   {
    return "İşlem başarılı";
   }
 protected static string protectedDeneme2()
 {
  return "İşlem başarılı";
 }
 protected static string protectedDeneme3()
 {
  return "İşlem başarılı";
 }
}
```




**Resim 8.2.** Protected Erişim Belirleyicilerinin Çeşitli Kullanımlarına Ait Sonuçlar

Resim 8.9.'da gösterildiği gibi "protected" ifadesinin farklı sınıfta kullanıldığı durumda metoda ait kısmi bir kısıtlama getirirken kendi sınıfından türetilen sınıflarda (class Erisim:Form1 ifadesinde belirtildiği gibi Erisim sınıfı Form1 sınıfından türetilmiştir.) "public" ile benzer sonuçları vermektedir. Örnekte belirtildiği gibi farklı olarak belirtilen Erisim sınıfı Form1 sınıfından türetilmek suretiyle metotlara ait her türlü erişime izin vermektedir. Metotlarda "static" yapısının metotları nasıl etkilediğine dair ayrıntılı bilgiyi ilerleyen kısımlarda belirtileceğinden detaylı açıklama yapılmamıştır.

## Private

Private erişim belirleyicisi ile oluşturulmuş metotlar başka sınıflar tarafından kullanılamaz.

"Private" anahtar kelimesi ile tanımlanan metotlar, sadece tanımlandığı sınıf tarafından kullanılabilirler. Bir başka ifadeyle "private" bir metot kendi sınıfı dışında erişilmemesi gereken ve sadece tanımlandığı sınıf ile kullanılması gereken bir yapı özelliğindedir. Bir metoda ait erişim belirleyici tanımlanmamış ise o metodun varsayılan erişim belirleyicisi "private"'tır. "Private" anahtar kelimesini tam olarak anlamak için örnek üzerinde görmek uygun olacaktır.

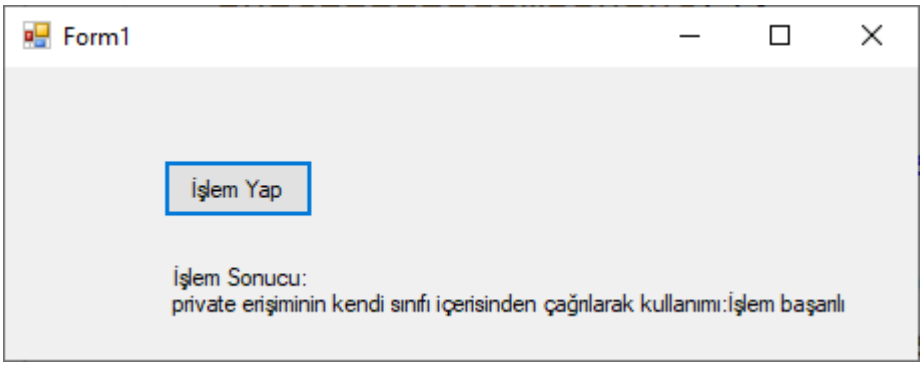
Örnek

- "Private" erişim belirleyicilerinin farklı kullanımlarına ait örnekler aşağıda belirtilmiştir.

```
class Erisim:Form1
 {
  public string MetotErisim()
   {
    return "İşlem Sonucu: "+"\n"
     + "private erişiminin kullanımı:" + protectedDeneme1() + "\n"
     + "private erişiminin statik ile kullanımı:" +       protectedDeneme1() + "\n"
     + "private erişiminin direk kullanımı:" + Form1.protectedDeneme3();
    }
 }

 public partial class Form1 : Form
 {
  public Form1()
   {
    InitializeComponent();
    }
 private void IslemYap_Click(object sender, EventArgs e)
 {
   Erisim erisim = new Erisim();
    IslemSonucu.Text = erisim.MetotErisim();
    IslemSonucu.Text ="İşlem Sonucu: "+"\n"
+ "private erişiminin kendi sınıfı içerisinden çağrılarak kullanımı:" + Form1.protectedDeneme3();

 }
  private string privateDeneme1()
   {
    return "İşlem başarılı";
   }
 private static string privateDeneme2()
 {
  return "İşlem başarılı";
 }
 private static string privateDeneme3()
 {
  return "İşlem başarılı";
  }
 }
```

Bir sınıf içerisinde erişim belirleyicilerinin kullanımında sayı olarak bir kısıtlama yoktur.

Yukarıdaki örnekte görüldüğü gibi “private” olarak tanımlanmış bir metot koruma düzeyinden dolayı hata(üstü çizili şekilde gösterilmiştir) vermektedir. Bu sebeple kodlar sağlıklı bir şekilde çalışmayacaktır. “Private” ifadesi başka sınıfların “private” olan metoda erişimini kısıtlamaktadır. Ancak yine örnekte gösterildiği gibi aynı sınıf içerisinde tanımlanmış “private” erişim belirleyicisine ait metotlar herhangi bir kısıtlamaya tabi tutulmadan kullanılabilmektedir. Örneğe ait sonuçlar Resim 8.10.’da gösterilmiştir.




**Resim 8.10.** "Private" Erişim Belirleyicilerinin Çeşitli Kullanımlarına Ait Sonuçlar

**Bireysel Etkinlik**

- Bu ünitede Nesne tabanlı programlamadan bahsettik. Acaba diğer programlama türlerinin işleyişi nasıl?
- Metotların kullanılmadığı bir yazılım düşünün. Böyle bir yazılımda en fazla kaç satır koda hakim olabilirdiniz?

## Özet

- Yazılımlar, insanların ihtiyaçlarını ve problemlerini teknoloji ile çözmek için kullanılan kodların bütünüdür. Özellikle veri ve bilginin önemli olduğu ve yoğun bir şekilde işlendiği bun yapılar, sistemlerin, projelerin ve otomasyon etkin ve doğru şekillerde gerçekleşmesi için çok önemlidir. Her geçen gün yaygınlığı ve etkinliği artan yazılımlar problemin karmaşıklığına bağlı olarak çok kolay veya zor algoritmalar içerdiğini söylemek mümkündür. Ancak algoritmalar zorlaştıkça yapıya hakimiyet ve müdahale de oldukça zorlaşmakta hatta imkansız hale gelmektedir. Tam bu esnada ortaya koyulan algoritmalar için kodları basitleştirmek, kodları azaltmak, güvenli ve düzenli kod yazımını sağlamak, isimlendirmeler ile anlaşılır hale getirmek, erişim belirleyicileri ile yetkilendirme işlemlerini rahat bir şekilde yapmak çok büyük bir önem arz etmektedir. Bunun için ise nesne tabanlı programlama, sınıf yapısı ve metotlar devreye girmektedir. Metotlar algoritmaların adım adım işlenebilmesi ve gerektiğinde kolay bir şekilde ten noktadan müdahalenin yapılabildiği özel yapılardır. Metotlar her ne kadar kalıplaşmış bir yapıda görünse de çok esnek kullanımlara sahiptirler. Metotlar amaçlarına ve işlevlerine göre çok çeşitli şekillerde kullanılabilmektedir. Metotları daha rahat kullanabilmek için yapılacak işlemleri tem bir amaç gözeterek gerçekleştirmek uygun olacaktır. Metotlardaki isimlendirmelerindeler de bu amaç doğrultusunda belirtildiği takdirde yazılımın anlaşılırlığı oldukça fazla olacaktır.
- Metotlar kullanım amaçlarına göre geri dönüş değerlerine sahip olabilmektedir. Yani metodun yapması gereken işlemi gerçekleştirmesinden sonra return ifadesi ile metodun sonucu olarak bir değer döndürmesi işlemidir. Metot eğer bir değer döndürmüyorsa void ifadesi kullanılarak metodun yalnızca metodun parantezleri içinde işlem yaptığı ve sonuç olarak herhangi bir değere sahip olmadığı anlamı çıkar.
- Metotların kullanım amaçlarına göre erişim seviyelerinin belirlenebilmesi, bu yapının hem daha güvenli hem kullanım amacına daha hizmet etmesi açısından önemlidir. Public, protected ve private erişim belirleyicileri metodun kullanım amacına göre değişkenlik göstermektedir. Public erişime sahip olan metotlara hem kendi sınıfları hem de başka sınıflardan erişmek mümkündür. Public ifadesi kullanılan metotlara diğer metot veya sınıflardan erişirken ya sınıftan bir nesne oluşturulur ya da statik ifadesi eklenerek hem doğrudan (aynı sınıfta ise) hem de sınıf üzerinden erişmek mümkün olmaktadır. Protected erişim belirleyicilere sahip bir metoda erişmek için aynı sınıfta olduğu zaman public erişim ile aynı iken farklı sınıftan erişmek isteyen sınıfın, kullanılmak istenen protected metoda sahip sınıftan türetilmesi gerekmektedir. Son olarak ise private erişim belirleyiciye sahip metotların kullanımından bahsetmek gerekmektedir. Bu kullanım ise metodun yalnızca kendi sınıfı tarafından kullanımına izin vermekte başka sınıfların erişimini ise kısıtlamaktadır. Metotların kullanımlarında dikkat edilmesi gereken diğer bir durum ise statik ifadesinin kullanımıdır. Bu ifade metodun sınıf üzerinde direk erişilebilir hale gelmesi yani herhangi bir nesne tanımlamadan kullanımına izin vermektedir.

# DEĞERLENDİRME SORULARI

1. Aşağıdakilerden hangisi bir metot tanımlanırken kullanılan ifadelerden değildir?
   a) Sınıf adı
   b) Dönüş tipi
   c) Metot adı
   d) Erişim seviyesi
   e) Giriş parametreleri

2. Aşağıdakilerden hangisi metotlara ait dönüş tiplerinden biridir?
   a) string
   b) class
   c) this
   d) return
   e) public

3. Aşağıdakilerden hangisi metot kelimesinin anlamına yakın ifade değildir?
   a) Fonksiyon
   b) Yöntem
   c) Sınıf
   d) Prosedür
   e) İşlem

4. Aşağıdakilerden hangisi nesne tabanlı programlamada metotların kullanılmasının faydalarından biri değildir?
   a) Kodların kolay anlaşılması
   b) Bir kere yazılmış bir metodu çok kere kullanabilme
   c) Algoritmanın kolay uygulanması
   d) Metot adı olarak çok kere Main kullanımı
   e) Doğru isimlendirme ile kolay kullanım

5. Aşağıdakilerden hangisi erişim belirleyicilerinin(seviyesi) kullanım amaçlarını en iyi şekilde tanımlar?
   a) Bir kere kullanılan bir metot bir daha kullanılamaz.
   b) Metoda ait bilgilerin görünmesini sağlar.
   c) Metotların kendi sınıfı içinde veya başka sınıf içerisinden erişilerek kullanılmasını yöneten ifadedir.
   d) Sınıfların metotların yapısına müdahalesini kolaylaştıran en önemli ifadedir.
   e) Erişim seviyesinin ifade edilmediği durumlarda erişim seviyesi protected olarak kabul edilir.

6. Aşağıdaki açıklamalardan hangisi bir metot için private erişim belirleyicilerinin en belirgin özelliğidir?
   a) Private ifadeler public ile sürekli karıştırılır.
   b) Private ile tanımlanmış bir metodun başka sınıflarda direk kullanımına izin vermez.
   c) Protected ile public erişim belirleyicilerinin ortak noktasıdır.
   d) Bir metodun kesinlikle kullanılmasına izin vermez.
   e) Bir sınıfta yalnızca bir adet private ifadesi kullanılır.

7. Bir metoda ait geri dönüş tipi hangisi olamaz?
   a) string
   b) int
   c) public
   d) double
   e) bool

8. "this" ifadesinin metotlardaki kullanımı ile ilgili en uygun açıklama aşağıdakilerden hangisidir?
   a) Bu ifade her yerel ifade için kullanılması uygundur.
   b) Bu ifade global ve yerel olarak aynı isimde kullanılan değişkenlerin kullanımı sağlar.
   c) Bu ifade sınıfın dışında tanımlı değişkenleri kolayca kullanmak için kullanılır.
   d) Bu ifade hem yerel hem de global(genel) olarak tanımlanmış bütün metotların başında kullanılır.
   e) Bu ifade ile farklı sınıfların bir arada kullanılması sağlanır.

9. AlanHesapla adında Protected erişim belirleyicisi (seviyesi) kullanılan bir metotun bulunduğu Deneme sınıfının Erisim adında başka bir sınıfta, türetme yöntemiyle kullanılması için kullanılan ifade aşağıdakilerden hangisidir?
   a) class Erisim,Deneme
   b) class Erisim:Deneme
   c) class Erisim>Deneme
   d) class Deneme:Erisim
   e) class Erisim(Deneme)

10. Aşağıdakilerden hangisi nesne tabanlı programlama da çok fazla kullanımı ve faydası olan aşırı yüklemeler(overloading) özelliğini açıklayan cümledir?
    a) Aşırı yüklenerek nesne tabanlı programın kasılmasını sağlayan özelliktir.
    b) Aynı sınıf içinde aynı isim ve aynı giriş parametreleri ile sonsuz sayıda metot tanımlanabilir.
    c) Aşırı yüklemeler özelliğinin kullanımı her geçen gün yok olmakta ve programın bozulmasına sebep olmaktadır.
    d) Aynı sınıf içinde aynı isimde fakat farklı giriş parametreleri ile tanımlanan metotlar farklı görevleri yapabilmektedir.
    e) Aşırı yüklemeler sadece metotlarda değil aynı zamanda değişkenlerde de kullanılabilir.

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

# YARARLANILAN KAYNAKLAR

Akbuğa, M.(2016). NESNE TABANLI PROGRAMLAMA-I Sınıflar. Erişim Adresi: https://docplayer.biz.tr/26099291-Unite-nesne-tabanli-programlama-i-okt-mustafa-akbuga-icindekiler-hedefler-siniflar.html. Erişim Tarihi:27.08.2021

Herbert, S. C. H. I. L. D. T. (2005). Herkes için C#. Alfa Yayınevi, 1.

Millî Eğitim Bakanlığı, (2017). Bilişim Teknolojileri-Metotlar. Ankara. Erişim Adresi: https://www.mku.edu.tr%2Ffiles%2F1874-8073aa76-3303-473a-9f28-2ac8face3cc8.pdf&usg=AOvVaw3FuYVZ0ibTblaoDez-sSpa, Erişim Tarihi: 27.08.2021

Static (C# Başvurusu) (2020, 25 Eylül), Erişim Adresi: https://docs.microsoft.com/tr-tr/dotnet/csharp/language-reference/keywords/static, Erişim Tarihi:18.08.2021

Ünal, G. (2014)., C# “static” Kullanımı, Erişim Adresi: http://kod5.org/c-static-kullanimi/, Erişim Tarihi:27.08.2021

Yanık, M. (2009). C# a Başlangıç Kitabı. Erişim Adresi: https://docplayer.biz.tr/1951106-C-a-baslangic-kitabi.html Erişim Tarihi:29.08.2021