

- Domain ,Web Sitesi,Web Sayfası,Hosting Kavramları.
- Web Tasarımı ve Web Programlama Kavramları.

ASP.NET(C#)

01 - Giriş

Bilgisayar Programı nedir?

- Bilgisayarlar verilen bir görevi yapmak için komutlara ihtiyaç duyarlar
- Bu komutlar dizisine **program** denir
- **Software/yazılım** ise bir programa veya programlar grubuna verilen addır.

1- Ekrana “Merhaba yaz”
2- “Lütfen iki rakam giriniz”
3- Rakamları oku
4- Rakamları çarp
5- Sonucu ekrana yaz

.....
.....
.....
.....
....
..
.

102- Programı bitir ve çıkış

Yazılım Türleri

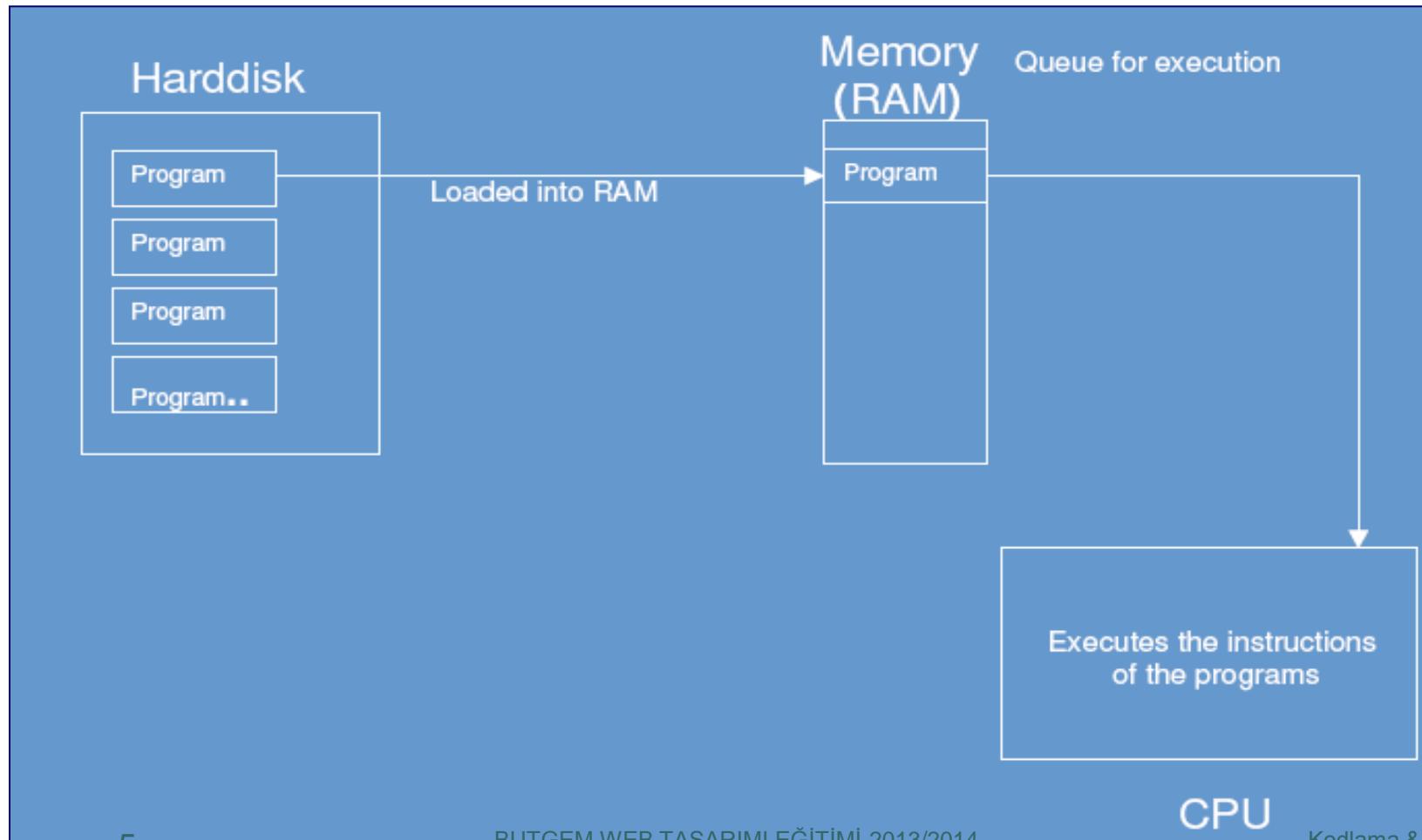
○ Sistem Yazılımı

- Cihazlarla direkt olarak etkileşimde olan programlar topluluğuna denir
- Ör. İşletim sistemleri (UNIX, Windows) veya sürücüler (ses kartı, ekran kartı, vs)

○ Uygulama Yazılımı

- Son kullanıcıların etkileşimde olduğu, çeşitli kolaylıklar sağlayan uygulamalar
- E.g. Kelime işlemciler, hesaplama, muhasebe, vb.

Programlar Nasıl Çalışır



Programlama Dilleri

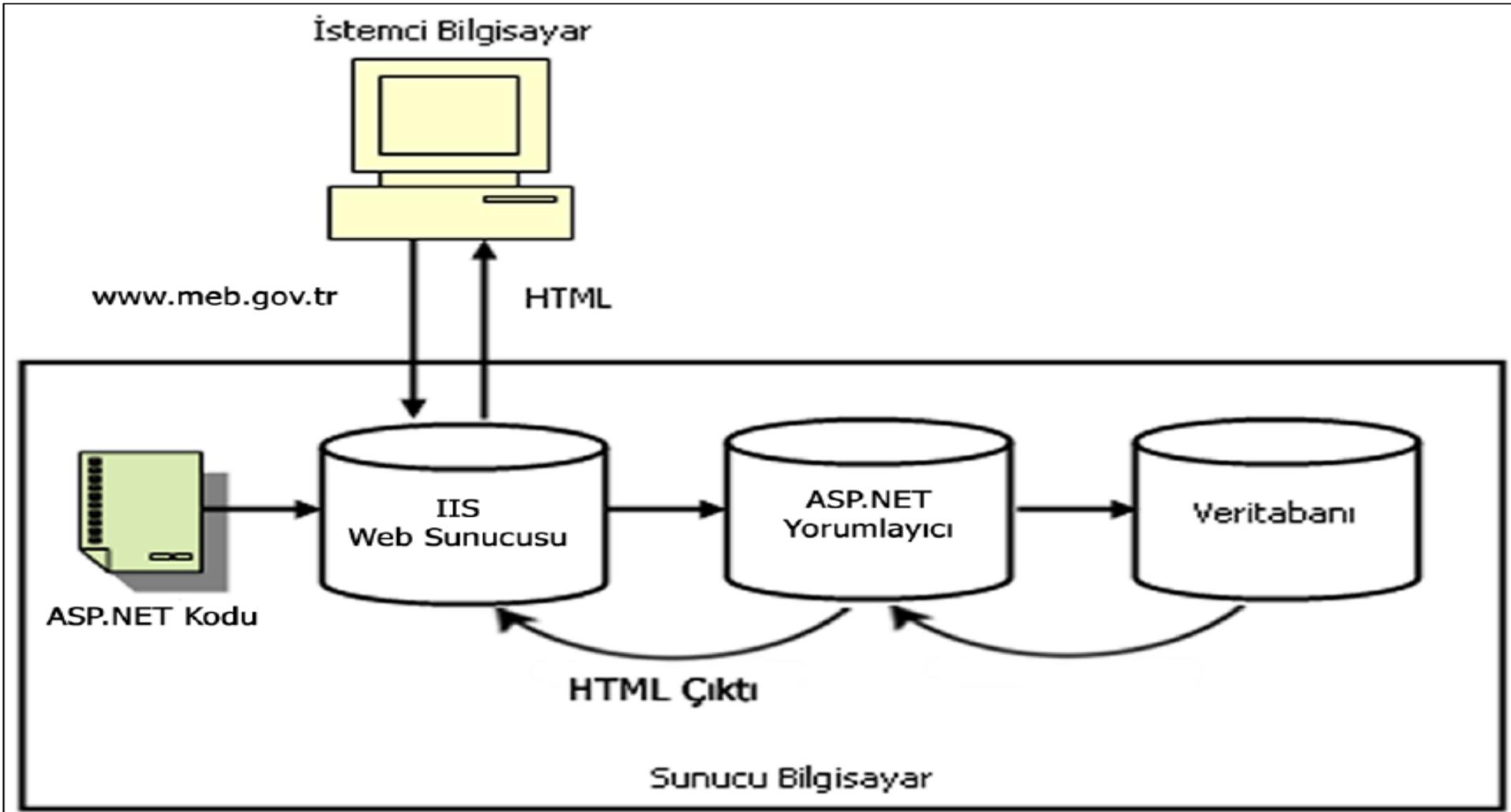
- Makine Dili (Machine language)
 - Bilgisayarların doğal dilidir
 - Makineye bağımlıdır
- Assembly Dilleri (Assembly language)
 - İngilizce benzeri ifadeler bilgisayar üzerinde uygulanır
 - Çevirici programlar (assemblers) makine diline uyarlar
- Yüksek Seviyeli Diller (High-level language)
 - İngilizceye çok daha yakın ifadeler kullanılır
 - Klasik matematik işlemleri içerir
 - Derleyici kodu makine diline çevirir
- Yorumlanan Diller (Interpreted Languages)
 - Derleme yapılmasına gerek olmadan, çalışma zamanında makine diline çevirim yapılır

ASP.NET

- ASP.net uygulamalarında sunucularda çalışan uygulamalardır.
- Kendi bilgisayarınızda çalıştırabilmek için. NET Framework'ü yüklemeniz gereklidir.
- .Net Framework işletim sisteminizi. Net uyumlu hale getirir.
- Redistributable (21 mb) ve tüm bileşenlere sahiptir.
- SDK (131 mb) tüm bileşenler yanında, dökümasyon ve örnek uygulamalarda vardır içerisinde.
- İndirmek için msdn.microsoft.com/netFramework
- Bilgisayarınızda .net Framework yüklemü
- Çalıştır→regedir→Hkey_local_Machine→Software—>Microsoft→Net Framework

Internet İşleyiş Modeli

- Internet sunucu istemci modeli ile çalışır. Bir istemci browser'inde site ismi yazılınlca isteğimizi bir sunucu karşılar ve bize cevap gönderiri.
- (request(istek) - response(cevap))
- Sunucu sadece html'den oluşan sayfaların bir kopyasını istemciye (hiçbir şey yapmadan) gönderiri. İstemci browser'ı bunu yorumlar. Aktif sayfalar hazırlamak için script'ler kullanılır.
- **2 Taraflı Script Vardır**
- a)İstemci taraflı: Sunucunun fonksiyonu yoktur sayfayı barındırır. Gönderir. Browser sayfayı yorumlarken bir script görürse gerekeni yapar.
- b)Sunucu taraflı: Sunucu tarafında işlenir. Sonuçlar html olarak istemciye gönderir.
- (E-Ticaret sitelerinin temeli).Asp.net,Asp,php,jsp



ASP.NET Ve Olay Yönlendirmeli Modeli

- İstek- cevap modelindeki en büyük problem bağlantısızlıktır. Yani internet ortamında sunucu ile istemci arasında direkt bağlantı yoktur. Bu TCP/IP mantığındaki anahtarlama tekniğinin sonucudur. Bu durum internetin ve zafiyet oluşturur.
- İşte ASP.Net bu bağlantısızlık işine yeni bir yorum getirerek olay yönlendirmeli(Bir programın çalışması için gereken eylemdir, örneğin tuşa tıklamak,olaylar programın akışını yönetir ve kontrol sağlar) hale gelmiştir.
- İşte bu olay mantığı önceki (ASP) sistemde tesis edilmesi çok zordu. Çünkü sunucu(sunucu) istemci bir istekte bulunmadıkça onun ne yaptığını bilmiyordu.

ASP.Net bu zorluğu aşmıştır.Nasılımı?

- ASP.Net yapısı istemci taraflı script ve ASP.net'e özgü yeni tekniklerden oluşur.İstemci taraflı scriptler(bir ajan gibi) istemcide olanı server'a bildirir.Server'da olaydan haberdar olur işlemi yapar.(Bu bir devrimdir) Çünkü programcı sadece kendi geliştireceği programa(web) bakar.
- ASP.net dilediğimiz bir dil ile (Vb,C,C#..) olay yönlendirmeli web uygulamaları geliştirmemizi sağlayan ve MSIL ve CLR ile platform bağımsızlığı ile taşınabilir dağıtılabılır web uygulamaları geliştirmemizi sağlar.

ASP ile ASP.Net Karşılaştırması

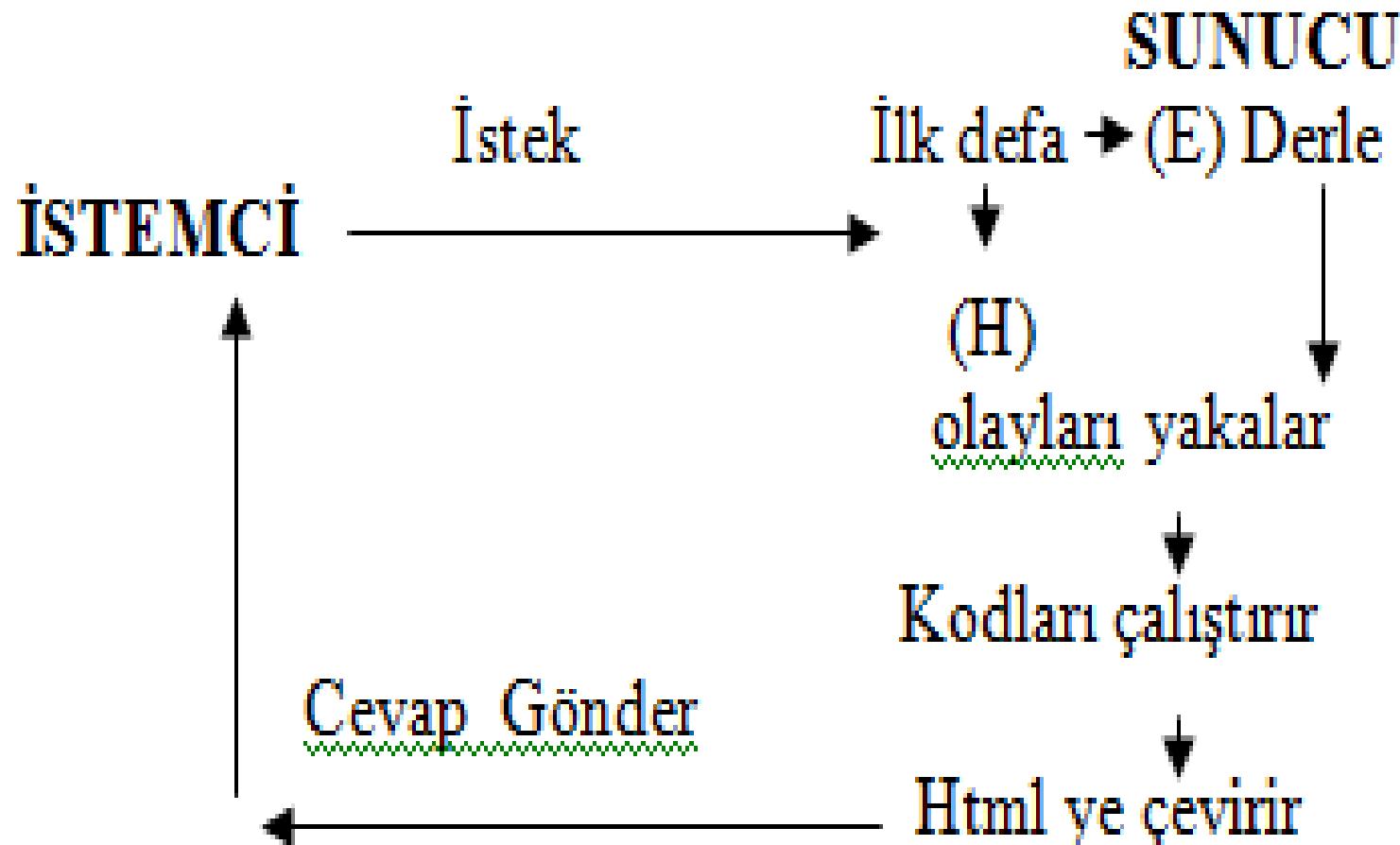
ASP

1. ASP kodu işler ve gönderir bağlantısızlık vardır.
2. ASP sayfaların bir takım işlemler için nesneleri o sunucuya yüklemeleri(register) yapmaları gereklidir.
3. ASP' de ise IIS yüklenebileceği platformlarda çalıştırılır.
4. ASP html form nesnelerini kullanırken (olay yöntemli değildir)
5. Form elemanları gönderildiği anda içlerindeki değerleri kaybederler.
6. ASP kodları yorumlanırken çalıştırılır bu yüzden performans düşüktür.

ASP.Net

1. ASP.Net istemci tarafından script ajanları ile bu zorluğu aşarak programa yoğunlaşmayı sağlar.
2. ASP.Net yapısında ise bu durum yoktur. .net framework yükleneninde MSIL sayesinde register işlemine gerek yoktur.
3. CLR ile ASP.net uygulamaları bütün platformlarda çalışırken.
4. ASP.net web kontrollerini kullanır bu sayede daha çok nesneye yönelik programlamaya ve performansa imkan verir (olay yöntemli).
5. ASP.net web formu yapısında Viewstate özelliği ile web kontrolleri son değerlerini unutmazlar.
6. ASP.net yapısında ise derlenerek kodlar çalışır buda hızı (performansı artırır).

ASP.Net Sayfasının Sunucu Üzerinde Çalışması



ASP.Net Sayfa Yapısı

```
<% page language="vb" codepage="1254"%> (Sayfa direktifi)  
<script runat ="server"> (Sunucuda çalışılacağını gösteri)
```

(Yordamların
bulunduğu
yer)

```
Sub page_load(obje as object,e as EventArgs)  
    Mesaj. Text="nasılsınız ilk deneme";  
End sub  
</script>
```

(Kod bildirim bloğu
ancak sunucuda
çalışır)

```
<html>  
<meta ...>  
<body>  
<asp:label ID="mesaj" runat="server"/>  
</body>  
</html>
```

(ASP. Net'e eşit web
kontrollün ve formu
barındırır)

- Microsoft Visual Studio ile Yeni Web Sitesi Açımak için → File → New Web Site → Programlama Dili olarak C# seçilir → Asp.Net Empty Web Site seçilir ve açılır.
- web.config sitemizin ayarlarını tutan dosyadır.
- Tüm web sayfaları istekde bulunulduğunda bu dosyaya bakarak yorumlanır.
- Web sitemiz üzerinde Sağtuş → Add new item → Web Form Seçilir.
- Asp.net Web Sayfa uzantıları **aspx** ‘tir.

C# ile Nesne Tabanlı Programlama

02 – Temel Kavramlar

Yorum Satırları

○ Başlama: //

- Yorum satırları atlanır, program yürütmesine katılmaz
- Kodu tanımlar ve dokumante eder
- Kodun okunabilirliğini arttırmır

○ Komut aralıkları: /* ... */

- /* Bu bir komut
aralığıdır */

Data & Değişkenler

○ Değişkenler

- Daha sonradan programın herhangi bir yerinde kullanılmak üzere türü ve boyutu belli olan kaplardır.
- Değişkenler hafızada geçici olarak(Ram) saklanan değerleri temsil eder.

○ Değer Tipleri

- Değişkenin hangi tipte değer tutacağını belirler
- Örn, bu değişken tam sayı mı tutar? Karakter mi tutar?

○ (Değişken adları)

- Çeşitli karakterler, rakamlar ve işaretler topluluğu (_) ve dollar işaretleri (\$) başlayabilir
- Rakamla başlayamaz ve boşluk içeremez
- Örn: welcome1, \$value, _value, button7
 - 7button GEÇERLİ DEĞİLDİR

○ C# case sensitive'dır (Büyük – Küçük harf farkeder)

- a1 ve A1 farklıdır.

Not: Değişkenler kalıcı değildir. Programdan çıktılığında değerler kaybolur. Kalıcı değerler için değişkenlerin değerleri diske yazılmalıdır.

Data & Değişkenler

Primitif Tipler (Primitive Data Types)

- Integers – Tam Sayılar
 - **byte** – 8 bit (-128 to +127)
 - **short** – 16 bit (-32768 to +32767)
 - **int** – 32 bit (-2 147 483 648 to 2 147 483 647) veya (- 2^{31} to $2^{31} - 1$)
 - **long** – 64 bit (çok daha büyük değerler)
- Characters - Karakterler
 - **char** – 16 bit, unicode, ASCII DEĞİLDİR!
- Floating point numbers (Ondalık Sayılar)
 - **float** – 4 bytes (- 3.4×10^{38} to $+3.4 \times 10^{38}$)
 - **double** – 8 bytes (- 1.7×10^{308} to 1.7×10^{308})
- Boolean data (evet/ hayır)
 - **boolean**
 - Yalnızca true veya false değeri alabilir

Data & Değişkenler

- Her değişken iki şeye sahip olmalı: **data tipi** ve **isim**.
- Kodun maksadına uygun, tanımlayıcı isimler verilmeli.

Örn:

```
alan = PI * radius * radius;  
a = p * r * r;
```

- Her değişkenin bir tipi olmalıdır

```
double salary;  
int vacationDays;
```

- Değişkenler kullanılmadan önce tanımlanmalıdır (initialized)

```
salary = 54000.0;
```

- Tek satırda yapılabilir

```
char yesChar = 'y' ;
```

- Bir harf ile başlamalı ve harf/rakam (Unicode) ile bitmeli

- Değişken isimlerinde türkçe karakter kullanılamaz.

Data & Değişkenler

- Programda kullanılmadan önce değişkenler tanımlanmalıdır.
- Değişkenin tanımlanması C#'da hafızada ne kadar yer ayrılması gerektiğini bildirir, ayrıca değişkenin ilk değerini hafızaya yazar.
- Değişken tanımlama çeşitleri:

Örn:

```
int sumGrades, remainder, r2d2c3po;  
double avg = 0.0, stdDev = 0.0;  
char initial3 = 'T';  
boolean completed = false;
```

Kutu Analojisi

```
int yas = 35;
```

Yas

35

Yas 35 değerini tutan bir kutudur, ve yalnızca tam sayı tutabilir

Data & Değişkenler

Referans Tipleri

- Kullanacağımız objeleri/nesneleri ifade eder, hafızanın HEAP kısmında tutulurlar
- Örn: System , Object(C# hersey bu sınıfından türer)
- *new* kelimesi kullanılarak initialize edilir
 - Örn: `Object o = new Object();`
- *String* bir referans tipidir fakat primitif tip gibi davranır
 - Örn: `String message = "Welcome to Java!";`
- Daha sonra *reference tipleri* ve *String* nesnesine ayrıca değinileceğiz.

Operatörler

- Aritmetik

+ , - , * , / , % , ++ , --

- Mantıksal

& , | ..

**&& ilk şarta bakar sağlanmaz(false) ise
diğerine bakmaz.**

|| ilk şart true ise diğer şarta bakmaz.

- Atama (Assignment)

= , += , -= , vb..

- Karşılaştırma - İlişkisel

< , <= , > , >= , == , !=

Operatörler - Arimetik

- Klasik aritmetik operatörler
 - Kullanım
 - * - çarpma
 - / - bölme
 - % - kalanı bulma
 - +, - toplama ve çıkarma
 - *Math.pow(sayı,üsdegeri)* belirtilen sayının üst alır.*Math.pow(3,3)* sonuç 27
 - *Math.sqrt(sayı)* →sayının karekökünü bulur.
 - Kalan operatörü % bölmeden kalan sayıyı döndürür
 $7 \% 5 \rightarrow 2$

Operatörler - Aritmetik

Operator(s)	Operation(s)	Order of evaluation (precedence)
*	Multiplication	Evaluated first. If there are several operators of this type, they are evaluated from left to right.
/	Division	
%	Remainder	
+	Addition	Evaluated next. If there are several operators of this type, they are evaluated from left to right.
-	Subtraction	

- Parantezler işlemin önceliğini belirtmek için kullanılır

Operatörler - Aritmetik

○ Kısa (Shorthand) Operatörler

- Tüm aritmetik operatörler için hesaplama ve atama aynı anda yapılabilir
- Örn:

```
int x = x + 7;
```

aynı

```
int x += 7;
```

```
int x = x / 3;
```

```
int x /= 3;
```

Operatörler – Aritmetik

○ Kısa (Shorthand) Operatörler

- ++ ve -- değeri 1 arttırmak veya azaltmak için kullanılabilir

```
int x = 2;  
int z = x++;  
int y = x--;
```



Post increment/decrement

Sonuçta x, z → 2, y → 3 olacaktır

```
int x = 2;  
int z = ++x;  
int y = --x;
```



Pre increment/decrement

Sonuçta x = 2 , z = 3 ve y=2

Atama (Assignment) Operatörü “=”

- Atama operatörünü primitif tiplerin referanslarını literal (değer) a işaret etmek için kullanırız.
- Primitif tipler başka bir primitif tipe atandığında değeri kopyalanır

```
int x = 12;
int y = x;
x++;
System.out.println(x);
System.out.println(y);
```

Çıktı:
13
12

NOTE : Bu referans tipleri için geçerli değildir (*devamı daha sonra*)

String Birleştirme (Concatenation)

- + operatörü kullanıldığında iki **Strings** yeni bir **String** nesnesine birleştirilerek dönüştürülür.
- + operatörü ile bir **String** ve başka bir tip birleştirildiğinde, diğer tipin değeri **String** e dönüştürülrerek birleştirilir
 - Diğer değer bir object ise, onun **toString** metodu çağrılarak **String** gösterimi elde edilir

Kontroller & Döngüler

- Eşitlik ve karşılaştırma operatörleri kullanılır:

Standard algebraic equality or relational operator	Java equality or relational operator	Sample Java condition	Meaning of Java condition
<i>Equality operators</i>			
=	==	x == y	x is equal to y
≠	!=	x != y	x is not equal to y
<i>Relational operators</i>			
>	>	x > y	x is greater than y
<	<	x < y	x is less than y
≥	≥	x ≥ y	x is greater than or equal to y
≤	≤	x ≤ y	x is less than or equal to y

Kontroller & Döngüler

- Mantıksal operatorler
 - Karmaşık kontrollerin yapılmasını sağlar
 - Basit kontrolleri birleştirir
- C# logical operators
 - **&& (VE) ilk şarta bakar sağlanmaz(false) ise diğerine bakmaz.**
 - **|| (VEYA) ilk şart true ise diğer şartta bakmaz.**
 - **& (boolean logical VE)**
 - **| (boolean logical inclusive VEYA)**
 - **^ (boolean logical exclusive VEYA)**
 - **! (logical DEĞİL)**

Kontroller & Döngüler

- o **If**

```
if condition
    statement1
End
```

isim Ahmet ise

Göster "Hello Ahmet"

C# yazılımı;

```
if (name=="Ahmet")
{
    Response.write("Hello Ahmet");
}
```

Süslü
parantezler etki
alanını (scope)
tanımlar

Kontroller & Döngüler

o If ..else

if condition

İsim Ahmet ise

statement1

Göster "Hello Ahmet"

else

değilse

statement2

Göster "Hello Nobody"

C# syntax

```
if (name=="Ahmet")
{
    Response.write("Hello Ahmet");
}
else
{
    Response.write("Hello Nobody");
}
```

Kontroller & Döngüler

- **If ..else if**

if condition

statement1

Else if

statement2

C# syntax

```
if (name=="Ahmet")
{
    Response.write("Hello Ahmet");
}

Else if ("name==kamil")
{
    Response.write("Hello kamil");
}

Else
{
    Response.write("Hello ismail");
}
```

Kontroller & Döngüler

o İç içe if İfadeleri

```
if (number < 10 && number >0)
{
    if (number > 5 )
    {
        Response.write("The Number is between 5 and 10");
    }
    else
    {
        Response.write("The Number is between 5 and 10");
    }
}
else
{
    Response.write("The number is not between 0 and 10");
}
```

35

Kontroller & Döngüler

○ Switch

```
String string ="";  
int i = 1;  
switch ( i )  
{  
    case 1:  
        string = "foo";  
        break;  
    case 2:  
        string = "bar";  
        break;  
    default:  
        string = "";  
        break;  
}  
  
Response.write(string);
```

Condition variable

switch ifadesi

char, byte, short

veya int değişkenlerini
Kontrol edebilir

→ Kontrol

Döngüden çıkış

Break yazılmazsa tüm
case ifadelerine girilir

Kontroller & Döngüler

- o While

```
int a = 0;  
int i = 0;  
While(i<100)  
{  
    a += i;  
    //    a = a + 1;  
    i++;  
}  
Response.write("Sum = "+ a);
```

Kontroller & Döngüler

○ Break

```
while (i<100)
{
    if( i == 3 )break;
    Response.write(""+ i );
    i++;
}
```

Döngüden çıkar
3 ten sonra devam etmeyecek,

○ Continue

```
while (i<100)
{
    i++;
    if( i == 50 )continue;
    Response.write(""+ i );
}
```

Kontrol doğru ise,
continue dan sonra
yer alan kısım
çalıştırılmayacak ve
dış döngünün bir
sonraki değeri ile
devam edilecek i++
yerdeğisse ne olur?

Kontroller & Döngüler

○ Do ... While

```
int a = 0;  
int i = 0;  
do  
{  
    a += i;  
//    a = a + 1;  
    i++;  
} While(i<100);  
  
Response.write("Sum = "+ a);
```

Kontroller & Döngüler

o for

for (*başlama* ; *kontrol* ; *arttırma*)

{

ifadeler

}

```
int a = 0;
for( int i = 0 ; i < 100 ; i++ )
{
    a += i;
}
Response.write("Sum = "+ a );
```

Diziler (Arrays)

- Değişkenler tekil değerleri tutmaya yarıyordu. Eğer bir değişken içinde aynı tipten birden fazla değer tutmak istersek dizileri kullanırız (**Arrays**)
- Değişkenler gibi diziler de kullanılmadan önce tanımlamalıyız
- Değişkenler gibi dizlerin de bir ismi ve tipi olmalıdır
- Değişkenlerden farklı olarak, dizilerin içereceği değer sayısı tanımlanmak zorundadır
- İlk elemanın indis numarası 0 ‘dır.[] içine yazılan eleman sayısıdır ve örn:[5] yazıyorsanız 5 elemanı vardır son indis 4 tür.

Dizileri Tanımlamak;

```
int[] myarray = new int [ 20 ];  
Char[] myStrings = new char [ 10 ];  
  
//Or  
int[] mynumbers = { 1 , 2 , 3 , 4 , 5 };
```

Diziler

- Dizi elemanlarına erişim;

```
int[] myarray = new int [ 6 ];  
myarray [ 0 ] = 10;  
myarray [ 1 ] = 20;  
myarray [ 2 ] = 30;  
myarray [ 3 ] = 40;  
myarray [ 4 ] = 50;  
myarray [ 5 ] = 60;
```

myarray

10	20	30	40	50	60
----	----	----	----	----	----

değerler

0

1

2

3

4

5

indexler

Diziler

○ Enhanced for İfadesi

- Sayaç kullanmadan bir dizinin veya kolleksiyonun içerisinde gezinilebilir
- Syntax

foreach (*parameter* in *arrayName*)
statement

○ Örn:

```
int[] numbers = {1,2,3,4,5,6,7,8,9};  
for (int i in numbers) {  
    Response.write ("rakam: " + i);  
}
```

Diziler

○ Çok boyutlu diziler

```
int[,] myarr= new int [ 6 ,3];  
myarray [ 0 ] [ 0 ] = 10;  
myarray [ 0 ] [ 1 ] = 20;  
myarray [ 0 ] [ 2 ] = 30;  
  
myarray [ 1 ] [ 0 ] = 11;  
myarray [ 1 ] [ 1 ] = 22;  
myarray [ 1 ] [ 2 ] = 33;  
  
myarray [ 2 ] [ 0 ] = 100;  
myarray [ 2 ] [ 1 ] = 200;  
myarray [ 2 ] [ 2 ] = 300;  
  
myarray [ 3 ] [ 0 ] = 111;  
myarray [ 3 ] [ 1 ] = 222;  
myarray [ 3 ] [ 2 ] = 333;  
  
myarray [ 5 ] [ 3 ] = 1;
```

myarr

10	20	30
11	22	33
100	200	300
111	222	333

??

Çok boyutlu diziler

```
String[ ][ ] aile = new String[3][ ];  
aile [0]=new String[]{"kamil","sebehattin","osman"};  
aile[1] = new String[] { "bart simsın", "cart simsın" };  
aile[2] = new String[] {"feyzullah","emruulah","kamurallah","haydi yallah" };  
  
for (int i=0;i<aile[0].Length;i++)  
{  
    listBox1.Items.Add (aile[0][i]);  
}
```

Primitif Tip Dönüşümü (Type Casting)

- Farklı veri tipindeki değişkenler birbirleri ile işlemle girdiklerinde, bu değişkenlerin tipini sonuç değişkeninin tipine dönüştürmemiz gerekebilir
- Örn:

long = int + int → implicit cast , int → long

Aşağıdakiler derlenmez!:

byte = int + int → explicit cast int → byte gerekiyor

int = double + double → explicit cast double → int gerekiyor

```
double d1 = 2344.2333;  
double d2 = 12233.4333;  
int sum = Convert.ToInt32(d1)
```



Büyük tipler küçük tiplere dönüştürüldüğünde (örn double --> int) büyük tipte değer kaybı gerçekleşebilir

String fonksiyonlar

- Equals → stringlerin eşitliğine bakar.
- Length --> String uzunluğunu verir.
- replace() → karakterleri istediğimiz karakterlerle değiştir.
- substring() → String içerisindeki istediğimiz kadarını alır
- ToLower () → Küçük harfe
- ToUpper () → Büyük harfe
- split() → belli bir karaktere göre böler
- trim() → Başındaki ve sonundaki boşlukları alır.
- indexof () → String içinde karakter arar bulursa yerinin numarasını yoksa -1 geri döndürür.

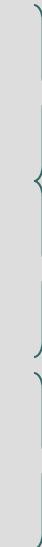
Exception Nedir?

- C#'da iki hata türü vardır:
 - Derleyici Hatası (Compiler Error) : Derleme öncesi yakalanan hatalar
 - Örn: karakter hataları, sentaks hatası, ...
 - Hata (Exception): Çalışma anında (runtime) gerçekleşen hatalar
 - Örn: Sistem hataları, cihaz hataları, dosya bulunamadı, dizi indeksi aşındı, ...

try - catch Bloğu

- Bir hata oluştuğunda hata “fırlatılır” (thrown)
- Hatayı yönetmekle sorumlu koda “exception handler” denir
- Hatayı yönetmek, bir hata oluştuğunda kodun ilgili yönetici tarafından işlenmesini sağlamaktır

```
try{  
    //hata yaratabilecek kod  
}  
  
catch (MyFirstException) {  
    //Hatayı yönetecek kod  
}  
  
catch (MySecondException) {  
    //Bu hatayı yönetecek kod  
}
```



Guarded Region

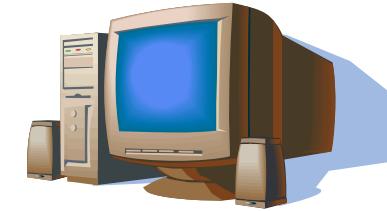
Exception Handler

Exception Handler

2 - Sınıf ve Nesnelere Giriş

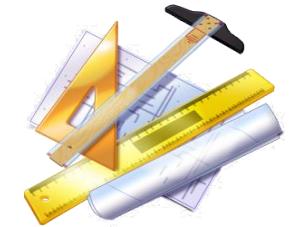
Nesne Tabanlı Programlama

- Hayatlarımız nesneler çevresinde kuruludur



- Bu nesneleri soyutlayarak yazılım projelerine yansıtırız
- Birden çok kez kullanım için nesneler soyutlanarak bilgisayar koduna dönüştürülür
- Oluşan soyut taslaklara sınıf (class) denir

Sınıf (Class) nedir?



- Araba Analojisi

- Mühendisler yeni bir araba üretmek için öncelikle proje planları oluşturur
- Benzin emisyonu, motorun nasıl çalıştığı gibi ayrıntılar bu planlara yansıtılır
- Planlar arabanın nasıl hareket edeceği, arabayı oluşturacak parçalar gibi birçok detayı içerir

- Herhangi bir sürücünün tüm bu detayları bilmesine gerek var mıdır?

- Hayır! Yalnızca ehliyetinin olması ve arabayı sürmeye bilmesi yeterlidir.

Nesne (Object) nedir?

- Nesneler sınıfların somutlaşmış halleridir
- Nesneleri durumu (state) ve davranış biçimleri vardır (behaviour)



Arabanın

-renk, hız, kapasite

-Hızlanmak ve

yavaşlamak için pedallar

} Durum (state)

} Davranış (behaviour)

- Sınıflar ise nesnelerin özellikleri ve davranışları ile ilgili ayrıntıları içerir Araba sınıfı.
-Frene basıldığında ne olur?



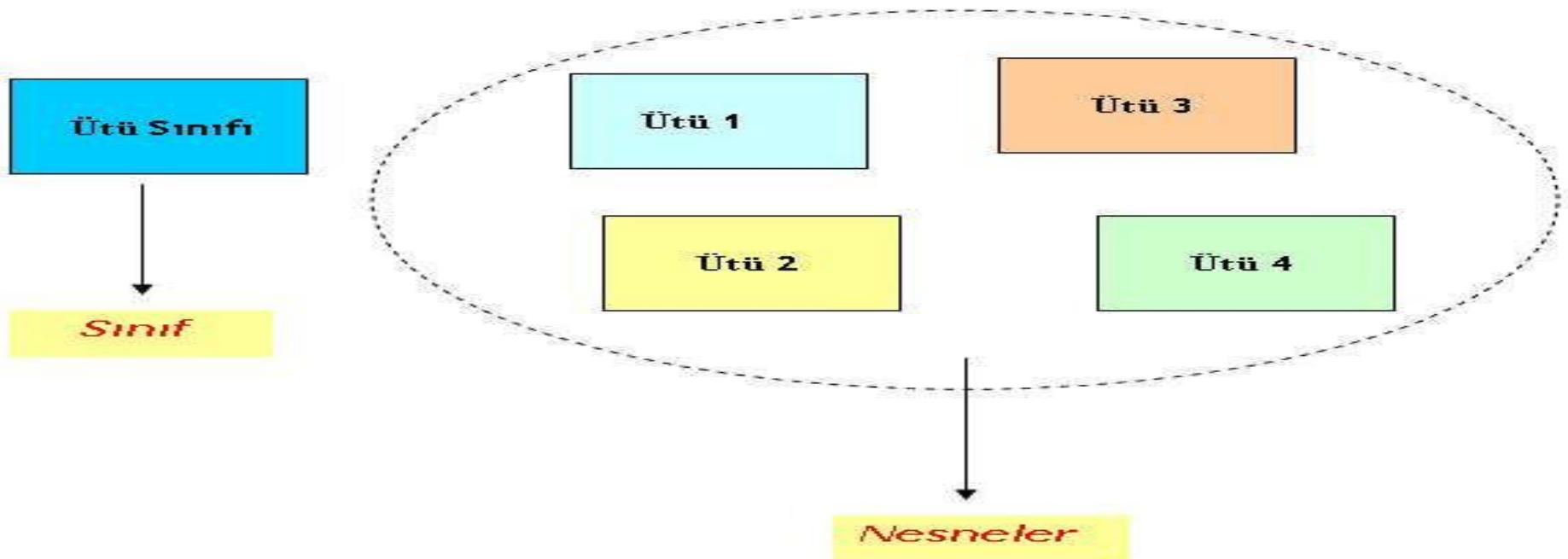
Nesnenin Durumu ve Davranışı

- **Durum (State):** Sınıfların bir – birçok özelliği olabilir
 - Somut değişkenler (instance variables) belirler
 - Nesneyle birlikte taşınır
- **Davranış (Behavior):** Sınıflar bir ya da birden çok metoda sahip olabilir
 - Metod program içindeki bir işi temsil eder
 - Görevlerin gerçekleştirileceği adımları tanımlar
 - Kullanıcıdan kompleks işlemleri gizler
 - Metodu çağrırmak, metodun bu işlemleri gerçekleştirmesini sağlar

Nesne (Object)

Mesela elimizde bir ütümüz olsun. Ütünün markası, modeli, rengi, çalıştığı elektrik voltajı, ne tür kumaşları ütleyebildiği bu ütüye ait özelliklerdir (durum).

Aynı zamanda ütümüza ısıtabiliriz, ütuleme içinde kullanabiliriz ve soğumaya bırakabiliriz. Bunlar ise ütünün fonksiyonlarıdır (metot-Davranış).

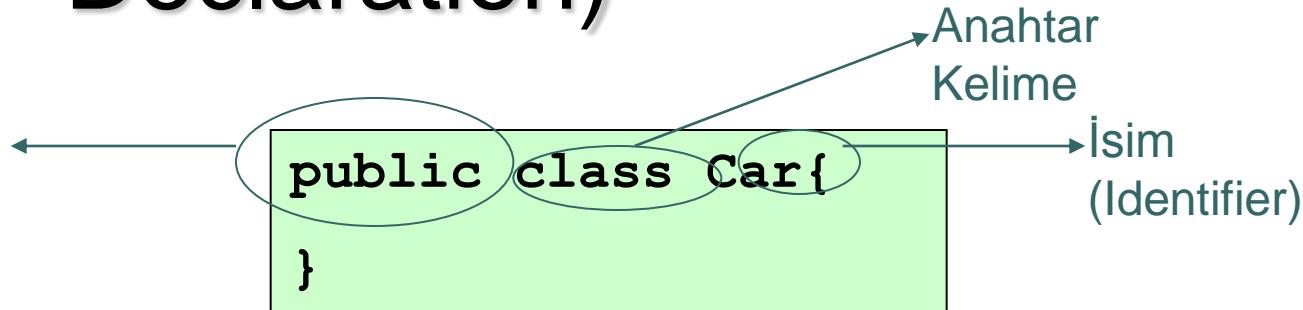


Nesne (Object)

- *Eğer ütü ile ilgili bir program yapmış olsak ve nesne tabanlı programlama tekniğini kullansak hemen bir ütü sınıfı (class) oluşturduk. Bu sınıfta ütüğe ait bilgiler (özellik-durum) ve ütü ile yapabileceğimiz işler (metot-davranışlar) bulunurdu.
- *O zaman nesne tabanlı programlamada bir sınıfta, sınıfa ait veriler ve bu verileri işleyip bir takım faydalı sonuçlar üreten fonksiyonlar / metotlar bulunur.
- *Dahası, biz bir tane ütü sınıfı tasarlarsak bu sınıftan istediğimiz sayıda değişik ütüler(Object veya instance) yapabiliriz.

Sınıf Tanımlaması (Class Declaration)

Erişim
(access
modifier)



Durum: *somit değişkenler*
(instance variables)

```
public class Car{  
  
    String color = "red";  
  
    int capacity = 5;  
  
}
```

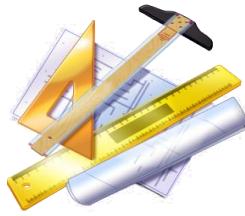
```
public class Car{  
  
    String color;  
  
    int capacity;  
  
    void moveForward() {}  
  
}
```

Sürücü arabanın
ilerlemesi için nesneye
moveForward() mesajını
gönderir.

Davranış: *metod*

Nesnenin bir Sınıftan Somutlaştırılması (Instantiation)

Sınıf: Car



Sınıf İsmi

Nesne:myCar



```
Car      myCar    =    new Car(); } myCar Nesnesi
myCar.accelerate();
```

identifier veya referans

Anahtar kelime

“.” ile metod çağrılmak

Metodlar

- Belli bir işi gerçekleştiren ifadeler topluluğudur
- Tüm metodların bir isimi vardır
- Fonksiyonlar C#'da metod olarak adlandırılır.
- Metodlar birden çok kez kullanılabilir.
- Her metod kendi değişkenlerine sahip olabilir (local variables), bu değişkenlere metod dışından erişilemez.

Metodlar

- Eğer metod bir değer döndürmüyorsa dönüş tipi olarak **void** belirtilir

```
public static void main ( String [ ] args )
{
    int result = multiplyTwoNumbers(28,1290);
    Response.write( "The result is = " + result );
}
```

Metodlar

- Parametreler metodlara veri göndermek için kullanılır ve metod içinde değişken gibi davranışlarırlar
- Dönüş tipleri metodlardan veri elde etmek için kullanılır
- *****C# Local değişkenlere tanımlandığı anda varsayılan(ilk) değerin verilmesini ister.*****
- **C#'da metod tanımlama:**

```
Access  
Modifier    Dönüş Tipi        Metod ismi        Parametreler  
↑           ↑                         ↑                         ↑  
public int multiplyTwoNumbers( int x , int y )  
{  
    int returnvalue = 0;      → Local Değişken  
    returnvalue = x * y;  
    return returnvalue;  
}
```

Değişken Uzunlukta Parametreler

- Parametre sayısı belli değildir

```
public double coklutoplam(double[] rakamlar)
{
    double t = 0;
    foreach (double sayı in rakamlar)
    {
        t += sayı;
    }
    return t;
}
```

Temel Kapsam (Scope) Kuralları

- Parametrelerin kapsamı metod gövdesidir (method body)
- Lokal değişkenin kapsamı tanımlandığı yerden içinde bulunduğu bloğu sonuna kadardır
- `for` döngüsünün başlığında tanımlanan lokal değişkenin kapsamı `for` başlığı ve `for` ifadesinin gövdesidir
- Bir sınıf metodunun kapsamı o sınıfın tümüdür

Constructor'lar

- Bir sınıfın değişkenlerini somutlaştmak için kullanılan metod
- C#’da her sınıfın bir constructor’ı bulunması gereklidir.
- Tanımlanmadığı durumda C# bir adet boş constructor tanımlar.
- Constructor’lar sınıf ismi ile aynı olmalıdır.
- Sınıf ismi ile **new** anahtar kelimesi ve sınıf isminden sonra parantezler eklenmesi ile çağrırlar
- Constructor’lar parametre alabilir fakat değer döndürmez
- Constructor’lar overload edilebilinir.
- Aslında bir nevi varsayılan değerleri belirttiğimiz metodlardır.

```
public class Car{  
    private String color;  
    private int capacity;  
  
    public Car(){  
        color = "Blue";  
        capacity = 5;  
    }  
}
```

Constructor, dönüş tipi
olmayan, sınıf ile aynı
isimde olan bir
metoddur

Gölgeleme (Shadowing)

- Bir metod içinde sınıf değişkenleri ile aynı isimde lokal değişkenler tanımlanırsa, sınıf değişkenleri lokal değişkenler tarafından gölgelenmiş olur.
- Gölgelenmeyi engellemek için metod içinden sınıf değişkenlerine erişmek için **this** anahtar kelimesi kullanılır

```
public class Car{  
    private String color;  
  
    public Car(String color){  
        this.color = color;  
    }  
    public void setColor(String color){  
        this.color = color;  
    }  
}
```

Primitif Tipler ve Referans Tipleri

- *Primitif tipler* (*boolean*, *byte*, *short*, *int*, *long*, *float*, *double*) yalnızca tek bir değer tutabilir
- Primitif instance değişkenleri varsayılan değerler ile atanır (*byte*, *short*, *int*, *long*, *float*, *double* → “0”, *boolean* → “false”)
- Diğer tüm tipler *reference tipleridir* (*yarattığımız nesneler gibi*)
- Referans tipleri hafızadaki nesneye erişmek için bir referans tutar
- Referans tipindeki instance değişkenleri *null* değeri ile atanır

Referans Tipi Atamaları

- Bir referans daha önce atanmış bir nesnenin referansına eşitlenirse, iki referans da hafızada aynı nesneye işaret eder.(Nesne olarak ayağa kaldırılmadığınız sürece)

```
Student stu1 = new Student();  
stu1.setName("Ahmet");  
Student stu2 = stu1;  
System.out.println(stu1.getName());  
stu2.setName("Mehmet");  
System.out.println(stu1.getName());
```

Çıktı:
Ahmet
Mehmet

stu1 ve stu2 hafızada aynı nesneye işaret eder. Böylece biri değiştiğinde diğerinin de değişir

Access Modifiers (Erişim) - public ve private

- **public/private** access modifier'ları metodlarda ve sınıf değişkenlerinde kullanılabilir
- Sınıflar **private** OLAMAZ
- **private** değişkenlere ve metodlara yalnızca tanımlandıkları sınıflardan erişilebilir
- **public** değişkenlere ve metodlara diğer sınıflardan da erişilebilir
- Private olarak tanımlanma sebebi dışarıdan erişimi engellemektir.(instance variables larımıza metodlar ile erişim sağlamaya çalışıcaz)

Access Modifiers - protected ve default

- protected access modifier:

- Sınıflar protected OLAMAZ
- protected değişken ve metodlara yalnızca bulundukları proje içinden erişilebilir

- default access modifier

- Eğer bir access mod. tanımlanmamışsa sınıflar, değişkenler ve metodlar private gibi davranışır

OOP Kuralı - Encapsulation

- Bir bileşenin - nesnenin veri yapılarının (*state*) dışarıdan saklanmasıdır
- Kullanıcılar bileşenin yalnızca ne yaptığıni bilmelidir, kullanıcılar işin nasıl yapıldığına bağımlı hale getirilemez
- Amaç değişim potansiyelinin karşılanabilmesidir
- Ayrıca nesnenin bütünlüğü korunmuş olur
- Örn: Sürücü yalnızca arabayı nasıl kullanacağını bilmelidir

Set - Get Metodları

- Bir nesnenin kullanıcısı nesnenin sınıfının private değişkenlerini değiştirmek için public metodlar kullanmalıdır .
- Bu metodlar lazım olduğunda kullanılır.

```
public class Car{  
    private String color;  
  
    public void setColor(String color) {  
        this.color = color;  
    }  
    public String getColor() {  
        return this.color;  
    }  
}
```

Set - Get Metodları –Otomatik Ekleme

- Hızlı bir şekilde sınıf değişkenlerimize (instance variables) set ve get metodları atayabilmek için değişken üzerinde sağ tuş
- →Refactor→Encapsulate Field seçilir.

Metodları Aşırı Yükleme (Overloading)

- Birden fazla metod aynı isimde fakat farklı parametre tipleri ve sıralamasıyla tanımlanabilir
- Derleyici hangi metodun çağırılması gereğine çağrıılan metodun parametre listesine bakarak karar verir
 - Bir metodun ismi, parametre tipleri ve sıralaması metodun imzasını belirler.**(parametre ismi bunu belirlemez.)**
- Metodların dönüş tipi metod overloading açısından bir etkiye sahip değildir.
 - Aşırı yüklenmiş metodlar farklı dönüş tiplerine sahip olabilir
 - Aynı imzaya sahip fakat farklı tipler döndüren metodlar compiler hatasına sebep olur

static değişkenler

- **static** değişkenler (veya sınıf değişkenleri)
 - Değişkenin bir kopyasının tüm sınıflar ve nesneler tarafından paylaşıldığı yapılardır
- Örn : **Math.max** , **Math.min** , **Math.sqrt**
- Kullanılması gereken durumlar:
 - Bir sınıfın tüm nesnelerinin aynı değişken değerini kullanması gereki̇ği durumlar
 - Bir değişkenin, bulunduğu sınıfından bir nesne oluşturulması gerekmeden erişilmesi gereki̇ği durumlar

static Metodlar

- **static** metodlar direk çağrırlabilinir:
Classısmi.metodısmi(parametreler)
- **Math** sınıfının tüm elemanları **static**'tir
 - örn: **Math.sqrt(900.0)**
- **static** metodlar bulundukları sınıfın **static** olmayan metod ve değişkenlerine erişemez

```
public static int makeSum(int a, int b) {  
    return a + b;  
}
```

Sabitler - Constants

- **Sabitler**

- Anahtar kelime: **const**
- Atandıktan sonra değiştirilemez
- Constructor'da veya atandığı anda bir değer atanması gereklidir

- **Math.PI** ve **Math.E**, **Math** sınıfının **final static** elemanlarıdır

3-Nesne Tabanlı Programlama Kavramları

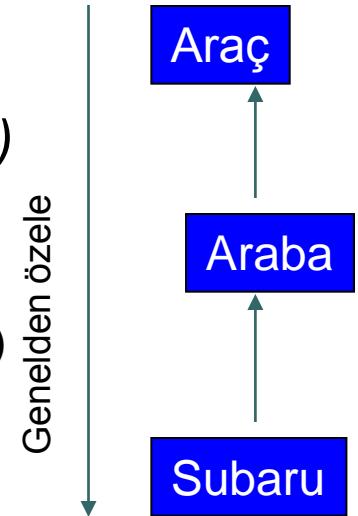
OOP(Object Oriented Programming) Kuralı - Composition

- Başka sınıflardan oluşmuş nesneler bir sınıfın değişkeni/ elemanı olabilir
- Bu *has-a* ilişkisi olarak da adlandırılabilir.
- Örn:Ogrenci→adres

```
public class Address{  
    /*...local variables, setters/getters, constructor...*/  
}  
  
public class Student{  
    private String name;  
    private Address _address;  
    public Student(String name) {  
        this.name = name;  
    }  
}
```

OOP Rule: Kalıtım(Inheritance)

- Kodun yeniden kullanılabilirinmesi
- Mevcut sınıfından yeni bir sınıf türetme
 - Mevcut sınıfın yapısı türeyen sınıfa aktarılır
 - Türeyen sınıfa yeni özellikler eklenebilir
- *Subclass extends superclass (Alt sınıf üst sınıfından türer)*
 - Subclass – Alt Sınıf
 - Özellikleri daha genişir
 - Davranışlar üst sınıfından aktarılır(örnek araç hareket met.)
 - Davranışlar değiştirilebilir
 - Yeni davranışlar eklenebilir
- *is-a* ilişkisi olarak da adlandırılır



```
public class Vehicle{/*Class body*/}
public class Car : Vehicle {/*Class body*/}
public class Subaru : Car{/*Class body*/}
```

Class Hierarchy – Sınıf Hiyerarşisi

- Direct superclass – Doğrudan Üst Sınıf
 - Üst sınıf bir seviye üstte(arac → araba)
- Indirect superclass – Dolaylı Üst Sınıf
 - Üst sınıf birden fazla seviyede üstte(arac → subaru arada araba var)
- Single inheritance – Tekil kalıtım
 - Yalnızca tek bir üst sınıfından türetme
- Multiple inheritance – Çoklu Kalıtım
 - Birden fazla sınıfından türetme
 - C# desteklememektedir!

Superclass & Subclass (Üst Sınıf – Alt Sınıf)

- Bir sınıfından oluşan nesne aynı zamanda üst sınıfından oluşan bir nesnedir
 - Örn: Dikdörtgen bir dörtgendir.
 - Dikdörtgen sınıfı dörtgen sınıfından türeyen bir sınıfır
 - Dörtgen: superclass – üst sınıf
 - Dikdörtgen: subclass – alt sınıf
- Superclass tipik olarak alt sınıflardan daha çok nesneyi temsil eder
 - Örn:
 - superclass: Araç
 - Arabalar, kamyonlar, yatlar bisikletler, ...
 - subclass: Araba
 - Daha dar bir araç kümlesi

Kalıtım Hiyerarşisi

- Kalıtım ilişkileri: ağaç hiperarşisi şeklindedir
- Her sınıf aşağıdakilerden biridir:
 - superclass
 - Diğer sınıflara eleman sağlar

VEYA

- subclass
 - Diğer sınıflardan eleman alır
- **private** değişkenlere
alt sınıflarda erişilemez
- **final** sınıflardan
türetme yapılamaz
- Her sınıf Object sınıfından türer

Alt sınıf üst sınıfın tüm public üyelerine ve kendi üyelerine sahip olur

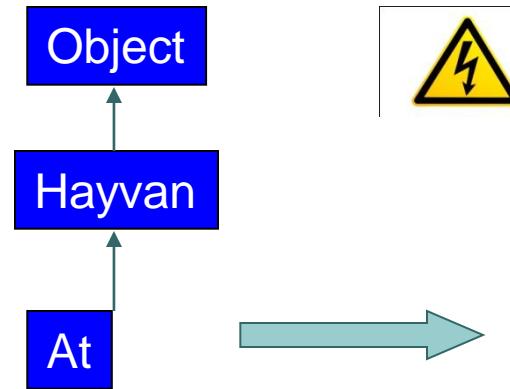
Kalıtım İlişkisinde Constructor'lar

- Constructor'lar alt sınıflara geçmez
- **Her alt sınıfın ilk görevi bir üst sınıfın constructor'ını çağrırmaktır.Yani alttan üste giderken eli boş gitmemeliyiz.**
- Eğer kodda üst sınıfın constructor'u çağrılmadıysa, C# kendiliğinden üst sınıfın default constructor'ını çağırır
- Üst sınıf constructor'u çağırmak için : **base(üst sınıf constructor parametresi)**
- Mevcut sınıfın başka bir constructor'ını çağırmak için: **this()**
- *Hepsinde default constructor varsa hiçbir sorun olmaz .Ama diğer constructorlarına erişemezler.*

Kalıtım ilişkisinde Constructor'lar

○ Constructor Chaining - Zincirleme

Nesne Hiyerarşisi:



Çalışmadan önce her constructor üst sınıfının constructor'ını kendiliğinden veya manuel olarak çağırır

Constructors sıralaması:

Main method çağrırlar: `new Horse()`

4. `Object()`

3. `Hayvan()` çağrırlar `:base()`

2. `At()` çağrırlar `:base()`

1. `main()` çağrırlar: `new Horse()`

initialization

Metodları Ezme : Overriding

- Bir metodun üst sınıfındaki çalışma şekli alt sınıfta değiştirilebilinir.
- Metot üst sınıfta virtual tanımlanmalıdır.

```
public class Animal{
    public virtual String makeNoise() {
        return "No Sound";
    }
}

public class Lion : Animal{
    public override String makeNoise() {
        return "Roarrr!";
    }
}
```

Metodları Ezme : Overriding

○ Overriding kuralları:

- Parametre listesi üst sınıfındaki metodla bire bir aynı olmalıdır
- Dönüş tipi üst sınıfındaki metodun dönüş tipiyle aynı veya onun bir alt sınıfı olmalıdır
- Access modifier daha kısıtlayıcı olamaz public → private olamaz (Fakat daha az kısıtlayıcı olabilir. protected → public olur)
- **private, final ve static** metodlar override yapılamaz (*fakat yeniden tanımlanabilir*)

Object Class

- C# her sınıf **Object** sınıfından türer.
- İyi tasarlanan sınıflar her zaman aşağıdaki metodları ezmeliidir
 - **equals** → gerçek nesnenin eşitliğini karşılaştırır.O yüzden kendimiz ezerek eşitlik durumuna göre true yada false gönderebiliriz.
 - **toString** → sınıfın kendisini **String** olarak temsil edebilmesine yarar.Bir nesne direkt çağrııldığından toString metodu çağrılır obje sınıfının,override edersek bizimki çağrılır.

Interface Uygulaması

- Bir interface 'ten türeyen bir sınıf üst sınıfın alt tipi (subtype) olur
- Subtype tüm interface metodlarını uygulamalıdır
- bir sınıf birden çok interface ten türeyebilir fakat birden çok sınıftan türeyemez

```
public class Circle : Drawable{  
    public void draw() { ... }  
    public void clear() { ... }  
}
```

- Interface'ler başka interface lerden türeyebilir fakat sınıflardan türeyemez

```
public interface : Interface1, Interface2{ }
```

Interface

- Her interface kendiliğinden `public` ve `abstract` olur
- Interface'lerin tüm metodları `public` ve `abstract` olur
- Tüm interface değişkenleri kendiliğinden `public`, `static` ve `final` olur

```
public interface Drawable {  
    public void draw();  
    public void clear();  
}
```

ArrayList

- Sıralama eleman ekleme sırasına göre gerçekleşir
- Aynı elemandan birden çok kez içerebilir
- Elemanlara indexleri aracılığıyla erişilir
- **Dizilerden farkı dizilerde eleman sayısı girilmeliydi burda ise gerek yok.**

```
ArrayList myList = new ArrayList();
    myList.add("Ali");
    myList.add("Veli");
    myList.add("Mehmet");

    String name = myList.get[2];
    System.out.println(name);
```

- ArrayList sınıfı kullanmak için sayfamızda using System.Collections ile kütüphane eklenmelidir.

4 -Asp.Net Nesneleri

Hazır Random Sınıfı

- Rastgele sayı üretmek için kullanılır.
- Random sayı=new Random();
- sayı.Next(); → int sınırlarında sayı üretir.
- sayı.Next(1,1000) → 1 ile 1000 arasında rastgele sayı üretir.

RESPONSE (CEVAP) NESNESİ

- Sunucudan istemciye veri aktarmaya yarayan nesnedir.
- **Write Metodu**: sayfaya sunucu tarafından bir şey yazdırma için kullanılır.
Response. write("ali gel")
- **Redirect Metodu**: Sayfanızdan başka bir sayfaya yönlendirme yapmayı sağlar. Response. Redirect("hata_sayfası. aspx")
- **Response.Clear() Metodu**: O ana kadar tampon bellekte saklanmış olan her şeyi siler. Yani o ana kadar çalıştırılmış olan kodların hiçbirinin sonucu gönderilmez.
- **Response.End() Metodu**: O zamana kadar tampon bellekte birikenleri gönderir. Bu komuttan sonrakileri ise göndermez.

REQUEST (İSTEK) NESNESİ:

- Response nesnesinin tam tersidir. Kullanıcıdan gelen verileri sunucuya ileter. Bu nesne, bir formdan veya bir sayfadan, başka bir sayfaya veri aktarımı için oldukça kullanışlıdır. ASP' de oldukça işlevsel olarak kullanılırken, .NET yapısında başka yordamlar kullanılır.
- Link ile Başka Bir Sayfaya Veri Götürme Ve Bunu Request İle Okuma
- Bunun için genellikle ASP.NET yapısında Response.Redirect() kullanılır ve yönlendirilen sayfada bilgi QueryString ile okunur.
- ÖRNEK Kullanım:
- Response.Redirect("oku.aspx?deger=1")
- Gideceği sayfa? Yanında götüreceği değer için değişken adı değişkenin değeri
- Oku.aspx sayfasında bu değer şu şekilde okunur:
- int a=Convert.ToInt32(Request.QueryString["deger"]);
Response.write(a) → sonuç 1'dir.

COOKIES (ÇEREZLER) :

- Ziyaretçinin bilgisayarında saklanan ve sunucunun daha sonradan bu bir takım bilgileri edinebileceği dosyalardır.
- Cookie'ler genelde bir ziyaretçi bir sayfayı ziyaret edince, ona ait bilgileri ziyaretçi bilgisayarında saklayıp daha sonraki ziyaretlerde onu tanımak için kullanılır.
- Çerezleri kullanabilmek için ziyaretçi bilgisayarında çerez alımı engellenmemiş olmalıdır.

Ziyaretçi Bilgisayarında Cookie Oluşturma

- HttpCookie sınıfından bir nesne oluşturulur. **HttpCookie cerez=new HttpCookie("bilgi");**
- Daha sonra nesneye alt anahtar bilgileri verilir.
cerez["ad"]="tuncay";
cerez["soyad"]="salı";
- Çerezin ölüm tarihi belirlenir.
cerez.Expires = DateTime.Now.AddYears(2);
- Response nesnesinin cookies özelliğinin add metodu ile cerez ziyaretçiye gönderilir.
Response.Cookies.Add(cerez);
- Çerezler windows 7 de
C:\Users\lastapex\AppData\Roaming\Microsoft\Windows\Cookies\Low
klasöründe tutulur.

Ziyaretçi Bilgisayarından Cookie Okuma

- `if (Request.Cookies["Bilgi"] != null)` ifadesi ile çerez var mı yok mu die kontrol edilir.
- `cerez = Request.Cookies["Bilgi"];` ile çerez okunarak `httpcookie` çerez nesnesine aktarılır.
- Daha sonra alt anahtarlar okunarak kullanılır.

`Response.Write(cerez["ad"] + cerez["soyad"]);`

SESSION(OTURUM)

- Oturum boyunca (yeni kişi browser'ını kapatana kadar) o kişiye ait bazı verileri tutan değişken tanımlamasıdır. Ve bu verileri sayfalar arası kullanabiliriz.
- Kullanımı:
Session[“değişken-ismi”]=değer
- Acaba sunucu bizi nasıl tanır ve izler?
- Server her ziyaretçiye 120 bitlik string bir ID verir. Bu da sessionID koleksiyonunda tutulur.
- Bunu da tutmak için ziyaretçi bilgisayarında bir cookie oluşturulur.
- **Session.timeout** : Session değişkeninin oturum sonlandığında iptal edildiğini söylemişтик. Bunun diğer bir şekli de tanımlanmış sürede sayfa üzerinde işlem yapılmadığında session'ın iptal olmasıdır.
- Timeout süresi; varsayılan olarak 20 dakikadır.

- ÖRNEK Kullanım:
- **Session.Timeout=10** yazarak bu süreyi 10 dk. Yapabiliriz.
- **Session.abandon** : Session.abandon yazarak istenilen yerde ani bir şekilde oturum sonlandırılabilir.
- **Session.Remove("değişkenadi");** Sesion'ı kapatmadan o değişkeni temizler.
- **Session.RemoveAll:** Session'ı kapatmadan tüm değişkenleri temizler.
- Normalde bir sayfada tanımladığımız bir değişken ve değeri ancak o sayfada kullanılabilir. Cookie ve session'ların kullanılmasında başlıca amaç ise; bu değişkenleri başka sayfalarda da kullanılabilir yapmaktır. Ancak burada en kullanışlı olan Session'lardır.

APPLICATION NESNESİ

- Bazı uygulamalarda tüm web uygulamamız için gerekli olacak değişkenlere ihtiyaç duyulabilir. Örneğin; sitemizde bir değişken tanımladık. Bunu bir kullanıcı 12 yaptı ise ve artık bu değerin tüm ziyaretçiler için 12 olmasını istiyorsak, bunu application nesnesi olarak tanımlarız.
- Kullanımı:
- Application[“degerismi”]=değeri şeklinde tanımlanarak değer atanır ve yine Application[“degerismi”] şeklinde okunur.
- Application içinde bilgiler string olarak tutulur.int’e çevirelecekse
- int sınıfının parse metodу kullanılır.**int.parse(Application[“değişken”])**

- **Application.Remove("değişkenadi");** Application kapatmadan o değişkeni temizler.
- **Application.RemoveAll:** Application'ı kapatmadan tüm değişkenleri temizler.
- Application sonlanması için Server'ın ya kapanması lazımdır.
- Veya Web server'ın stop veya restart edilmesi gereklidir.

SERVER NESNESİ

- **Server.HtmlEncode**
- Response ile gönderilen HTML taglerini özel olarak kodlar ve yorumlanmamasını sağlar. Yani olduğu gibi yazdırır.
- Örnek:
- Response.write("tuncay <hr>") ekrana tuncay yazıp çizgi çizerken,
- Response.write(server.htmlencode("tuncay<hr>")) ekrana direkt tuncay<hr> yazar.
- **Server.HtmlDecode**
- HtmlEncode' un tersini yapar. Yani HTML kodlarını tekrar yorumlatır.
- Örnek:
- response.write(server.htmldecode(server.htmlencode("tuncay<hr>")))

Server.URLEncode

- Bir sayfadan başka sayfaya, örneğin değer taşırken & ifadesi bir değişkeni gösterir.
- Response.Redirect("serverurl.aspx?adi=Tuncay&sali") ifadesini gönderdiğimiz değer serverurl.aspx sayfasında Request.QueryString["adi"] ile okunduğunda, tuncay verisini gösterir. Çünkü & ifadesinden sonrakini de ayrı bir değişken olarak düşünür. İşte böyle url ile gönderilen özel ifadeleri (& gibi) salt veri olarak almak için, Server.URLEncode kullanılır.
- Örnek Kullanım:
- response.redirect("serverurl.aspx?adi=" + server.UrlEncode("tuncay&sali"))
- Request.QueryString["adi"] ile okunduğunda, tuncay&sali değeri alınır.

- **Server.MapPath**

Sunucudaki bir dosyanın sunucuda bulunduğu fiziksel yerin adresini verir.

Örn:Response.Write(Server.MapPath("cerez.aspx"));

Sonuç=E:\aspnetuygulamaları\WebSite1\cerez.aspx

- **Server.Transfer**

Server nesnesinin sayfayı başka bir sayfaya yönlendirmesini sağlar.

1.Sayfada bu komutla 2. sayfaya yönlendirildiğinde 1.sayfanın işletimi durur, 2.sayfaya gidilip oradaki işlemler yapılır. 1.sayfaya dönülmez. Adres çubuğu ise, 1.sayfa adresi görünür.

- **Server.Execute**

Server.transfer 'den farklı olarak; yönlendirilen 2.sayfa işlemleri yapıldıktan sonra 1.sayfaya geri dönülerek, işletimine kaldığı yerden devam edilir.

Hazır Klasör ve Dosya Komutları

- Bu komutları kullanabilmek için System.IO namespace sayfamıza eklemeliyiz.

Directory Sınıfı:Klasörler üzerinde işlemler yapmamızı sağlar.

- **CreateDirectory** metodu=Belirtilen yola klasör oluşturur.

Örn: String yol = Server.MapPath(""); → sitenin kök dizinini bulur.

```
Directory.CreateDirectory(yol + "/resim");
```

- **Delete**:Belirtilen yoldaki içi boş klasörü siler.

Örn: Directory.Delete(yol + "/resim");

- **GetCreationTime**:Klasörün oluşturulma zamanını bize verir.

Örn: Response.Write(Directory.GetCreationTime(yol + "/ismail"))

- **Exists**:Belirtilen yolda klasörün olup olmadığına bakar. Varsa geriye true yoksa false döndürür.

Örn: if (Directory.Exists(yol + "/utku") == true)

- **GetDirectories**:Belirtilen yoldaki klasör adlarını dizi olarak getirir.

Örn: String[] klasorler = Directory.GetDirectories(yol);

- **Getfiles**:Belirtilen yoldaki dosya adlarını dizi olarak getirir.

File Sınıfı:Dosyalar üzerinde işlemler yapmamızı sağlar.

- **Copy** metodu=Belirtilen yoldaki dosyayı belirtilen yola belirtilen isimle kopyalar.

Örn: String yol = Server.MapPath(""); → sitenin kök dizinini bulur.

File.Copy(yol + "/baris/yunus.doc", yolum + "/emrah/balina.doc");

- **Delete**:Belirtilen yoldaki dosyayı siler.

Örn: File.Delete(yol + "/resim/lale.jpg");

- **Move**:Bir dosyayı bir yerden başka yere taşımamızı sağlar.

Örn: file.Move(yol + "/img/4.jpg", yol + "/resim/4.jpg");

- **Exists**:Belirtilen yolda dosyanın olup olmadığını bakar. Varsa geriye true yoksa false döndürür.

Örn: if (File.Exists(yol + "/utku/lale.jpg") == true)

WEB FORM ELEMANLARI

- <asp: kontrolismi id:” ismi” runat=“server” özellikleri olayları />
- Sunucu kontrolleri 4'e ayrılır.
- HTML Kontrolleri
- Bildiğimiz html kontrol taglerinin sonuna runat= “server” yazarak sunucu kontrolü haline getirebiliriz. Ancak web form elemanları kadar fonksiyonel ve yetenekli değildirler.

Bu şekilde kullanılabilirinmesinin sebebi, eski ASP ile yapılan sayfaların yeniden dizayn edilmesi yerine yeni sisteme (.NET) hızlı bir şekilde adapte edilmesidir.

- **Web Form Elemanları**
- **Geçerlilik Kontrolleri**
- Kullanıcıların girdiği verilerin doğru olup olmadığını kontrol ederler.
- **Kullanıcı Kontrolleri**
- Programcı tarafından oluşturulan özellik ve metodlara, olaylara sahiptirler.

WEB FORMLARININ POSTALANMASI

- **Biriktirilerek**

Bunun için formumuza bir buton ekliyoruz. Kendisine hiçbir olay yönlendirmesi yapmıyoruz.

- **Hemen (Auto-postback=True)**

Web form kontrolünden ayrıldığımızda ya da içeriği değiştiğinde eğer kontrolün auto-postback özelliği True ise, web formu çalıştırılmak üzere server'a postalanır.

Auto post-back nasıl mı yapılıyor? Bunun için server kullanıcı taraflı bir script olan javascript kullanıyor. Ancak bunun için bizi bir şey yapmamız gerekmiyor. Bu javascript kodlarını çalışma anında kendisi oluşturuyor.

ISPOSTBACK ÖZELLİĞİ(page.ispostback)

- Sayfanın ilk yüklenmesi durumu ile daha sonradan postback (geri bildirim) yapılması durumundaki işlemler bu özellik ile ayrılır.
- Eğer sayfa postback yapılarak karşımıza geldi ise; ispostback, true değeri üretir.
- Ekranda refresh dahi yapılsa, sayfanın postback yapıldığını algılar.

VIEW STATE (Mevcut Durumu Koruma)

- Form elemanlarını gönderdikten sonra geriye döndüğümüzde bunların korunduğunu görürüz. ASP.NET'te bunun için bir çok satır kod yazmamıza gerek yoktur. Bu işlemi bizim için server yapar. Ziyaretçi bilgisayarında her kontrol için gizli nesneler oluşturur ve bunlar içinde değerleri tutar. Bunları ziyaretçi bilgisayarında tuttuğu için server da yorulmaz.

View State Önemli Noktalar

- View State değerleri ziyaretçide saklanır. Sunucuda hiç yer kaplamaz.
- Sayfa kendisini postback(geri dönüşüm) yapmayacak ise, viewstate kullanmak gereksizdir.
- Kontrol elemanlarında bir olay yakalanmayacaksız durum bilgisi tutmanın anlamı yoktur.

- Kontrol elemanına dışarıdan bilgi girilebilir durumda olmayabilir, o zaman viewstate kullanımına gerek yoktur.(Örneğin DataGrid)
- İşte böyle durumlarda viewstate kullanmak sayfa boyutunu gereksiz büyütür, bu da hızı etkiler.

View State Yönetimi

- Eğer Web uygulamamızın tümünde viewstate istemiyorsak; Web.config dosyasına;

Web.config 'de system.Web tagları arasına yazılır.

```
<pages enableviewstate="false"> </pages>
```

- Sadece o sayfada kullanmak istemiyorsak; sayfa direktifi kısmında;
`<%@page Enableviewstate="false" %>`
- Sadece kontrolde istemiyorsak; kontrolde enableviewstate="false" yapılmalıdır.

WEB SUNUCU KONTROLLERİ

AdRotator

Reklam bannerlarını yöneten kontroldür. Rastgele bir banner(resim) çıkarır.
Banner'a linkte eklenir.

- Bu kontrolün alacağı bilgiler bir xml dosyasında bulunur.
- XML dosya içeriği:

<Advertisements>

<Ad>

<Imageurl> ~\resim\1.jpg </Imageurl>

<Navigateurl> Tıklanınca gidilecek link </Navigateurl>

<Alternate Text>Üzerine gelince çıkacak yazı</Alternate Text>

<Keyword> Kategoriye ait olduğunu gösterir. Filtrelemede kullanılır.</Keyword>

<Impressions> Reklamlar arasında gösterilme sıklığı oranı </Impressions>

</Ad>

- Her <Ad>....</Ad> bloğu bir reklamı ifade eder.
- Reklamları okutmak için Adrotator nesnemizin advertisementtitle özelliğine = “Hazırladığımız xml dosyamızın ismi” verilir.

- **Keyword Filter özelliği ilede**

Banner xml dosyamızdan istediklerimizi göstermek için adrotator nesnesinin KeywordFilter özelliğine xml tanımlarken yaptığımız grplardan (keyword) birini yazdığımızda sadece o gruba ait reklamlar gelir.

CHECKBOX ve CHECKBOXLIST

Birden fazla seçenek aynı anda seçmek ve tek seçimler yapmak için kullanılır.

Eğer tek bir checkbox kullanıldı ise; seçili olup olmadığını anlamak için, Checkbox1.checked=True ise seçildir; False ise seçili değildir.

- Eğer checkboxlist kullanıldı ise;

Checkboxlist.items[liste sırası].selected ile listede seçili olan öğe bulunabilir.

items.count : Checklist'te kaç adet checkbox olduğunu bulur.

Repeatdirection : Bu özellik ile checkboxlist'in konumu yatay veya dikey olarak ayarlanır. Repeatdirection.vertical Dikey olur.

Temel olayı bu nesnelerin seçenekte değişim olup olmadığına bakar
Checkbox → checkedchanged checkboxlist → selectedindexchanged

DROPODOWNLİST-LİSTBOX

Açılan veya liste kutularıdır.

Seçili elemanın değerini: **Selecteditem.value** , text değerininide **Selecteditem.text** özelliği ile elde edebilir.

- items.add("yeni değer") eleman eklenir
- removeat(indexno) verilen index nolu elemanı siler.
- items.remove(string)belirtilen string elemanı listede bulur ve siler.
- Items.findbytext items.findbyvalue listedeki elemanı bulur ve getirir.
- Seçimlerde değişim olması olayı selectedindexchanged
- Selection mode: Multiple CTRL ile biren fazla seçim

RADIO BUTON-RADIOLIST

- Özellikleri checkbox-checkboxlist'e benzer.Ancak Radio butonlarda sadece bir seçenek seçilmesi gereken durumlarda kullanıldığı için tek tek radio buton tanımlamalarında groupname aynı verilmelidir.yoksa checkbox işleri görür ve hepsi seçilebilir .Bu durum radiobuttonlist'te yoktur.

Bir radiobutton'un seçili olduğu. **Radio1.checked=true**

Radiobuttonlist'te ise seçili radiobuttonun ile anlaşılır değeri (value) veya text özelliği şöyle elde edilir. **Radiobuttonlist.selecteditem.value**

- selectedindex** Seçili elemanın index numarası alınabilir veya ilk anda istenilen eleman seçtirilebilir.
- ÖR/ radiolist1.selectedIndex=2
- items(index).text= istenilen index nolu eleman değeri verir.Radiolist1.items(1).text
- items.count radiolist'teki radio buton sayımızı verir

TEXTBOX

Bilgi girişi için kullanılan nesnedir.

Text mode özelliği değiştirilerek

- Singleline Tek satır verir
- Multiline Çok satırlı verir
- Password Şifre girişi oluşturur
- Readonly → Sadece okunabilir yazılamaz.
- Autopostback → textchanged
- Tooltip text → Üzerinde iken çıkışacak açıklama
- Maxlength → Girilebilecek max. Karakter sayısı.
- Tabindex → tab tuşunda sırası.

BUTTON- LABEL

- Button Üzerine tıklandığında gerekli prosedüre çalıştırarak işlemler yapılır.
- Label etiket ve mesaj gibi işlemleri(görüntüleme) için kullanılır.

Normalde bir butonun görevi tıklandığında içinde bulunduğu sayfaya kendi postback yapmaktadır. İstenirse tıklandığında başka sayfada postback olabilir. Postback url özelliği ile bu ayarlanabilir.

HYPERLINK

HTML linki sunucu kontrolü olmalıdır. NavigateUrl özelliğine link adresini istersek link arayüzünü resimde yapabiliriz. Bu da imageurl kısmında belirtilir.

PANEL

Diğer kontrol elemanlarını çerçeve içinde barındırıp bu kontroller group halinde görünmez veya görünür yapılabılır.

IMAGE BUTTON VE IMAGEMAP

Buton kontrolü gibi sunucuya postalamaya yapan kontroldür. Ancak görüntü olarak resim kullanılabilir.

- Imgurl özelliğine verdığınız (resim) burada görülür.
- Borderwith, Borderstyle ve color ile resim etrafına çerçeve konulabilir.

Image Map

ise bir resim'in istediğimiz kısımlarına bölerek istediğimiz yerlere yönlendirebiliriz.

- Hotspot özelliği ile istenilen biçimde daire, dikdörtgen koordinatlara bölünür. hotspot mode ile o kısma tıklanınca ne yapacağı seçilir.
- Postback olabilir
- Negative Başka sayfaya link olarak çalışabilir.

LINK BUTON

Buton link şeklinde görünenidir ve Hyperlink'ten farkı link buton postalamaya yaparken, hyperlink ise bir adrese sadece köprü için kullanılır.

BULLETED LIST

Madde imleri , maddeler halinde verileri sunmak için kullanılır.
Bullet style ile çıkacak simge istenirse Style → Custom image seçilirse bizim seçtiğimiz image olabilir.

Displaymode Maddeler halinde sunulan verilen

Text

Hyperlink

Linkbuton

olması sağlanabilir.

PLACE HOLDER

- Panel nesnesine benzer ancak çalışma anında içerisinde nesne eklenebilir. Panel'den en büyük farkı budur ve görünür bir arayüz birimi yoktur.
- ÖR/ Textbox metinkutusu =new textbox();
Metinkutu.id="text1"
Placeholder1.controls.add(metinkutu)
Placeholder1.controls.add(new literalcontrol("isim"));

TABLE

Html tag'i yazmadan uygulamamıza kodlarda istersek table oluşturabiliriz.

- Table'a satır eklemek için(çalışma anında)

Table1.rows.add(new table row)

- Satır içine hücre ve bilgi eklenecek istersek

Satir.cells.add(new table cell)

- **Borderwidth, Style ve color**:Tablonun ana çerçevesinin ayarlar.

- **Grid lines** : Both seçilerek tüm hücreler çerçeve(yardımcı çizgi) içine alınır. Vertical,Horizontal satırlar grid içinde (yardımcı çizgi) olur.

- Eğer hücrelere çerçeve içine almak ise:

Border width, Border Style, Border color vermek gereklidir

- ÖR/

```
For (int i=1;i<=3;i++)
```

```
{
```

```
Tablerow satir =new Tablerow();
```

```
For (int i=1;i<=3;i++)
```

```
Tablecell hucre=new Tablecell();
```

```
Hucre.text ="be gev"
```

```
Satır.cells.add(hucre)
```

```
}
```

```
Table1.Rows.add(satir);
```

```
}
```