**DERS-1: GENEL TANIMLAR**

**C# Nedir?**

- Microsoft tarafından .NET çatısı altında geliştirilen ve gelişen modern bir açık kaynak programlama

dilidir.

- Object oriented programming'i destekler.

- Orta seviyeli bir dildir. (c,c++,java,ADA)

- C#, C++'ın bir üst versiyonudur. ++ zamanla #'a dönüşmüştür.

- Java'dan etkilenmiştir.

- C# ile WEB uygulamaları ve servisleri, Mobil, Konsol, Oyun, Servis mimarileri, ERP, DLL-... her şey geliştirilebilir.

(Veri madenciliğinde Python tercih edilir)

- Derlenen bir programlama dilidir.

**.NET**, Microsoft'un developer'lar için sunduğu bir çatıdır. Bu çatının altında kişinin geliştirmek istediği uygulamaya lazım her şey bulunur.

**.NET Framework** ilk çıkmıştır. Sadece Windows'a yönelik çözümler getirir. Microsoft bağımlıdır. Open source değildir.

**.NET Core** ise cross platform'dur. Çeşitli işletim sistemlerinden uygulamaları destekler. Modüler'dir. Open source'dır.

C# .NET Core'un bir alt koludur.

**Compiler Nedir?**

- Yazdığımız kodun makinenin anlayabileceği koda derlenmesi lazım. Compiler arada bir katmandır. Bu işi yapar.

- .exe de bu derleme sonucunda programı açmamızı sağlayan yazılım uzantısıdır. (DLL de benzer şekilde)

deneme dosyasını derlemek için:

using System;

namespace ConsoleApp1

{

class program

{

static void Main.....

{

Console.WriteLine("Merhaba Dünya");

Console.Read();

}

}

Developer command prompt ile dosyaları derleyebiliriz.

cd C:.... ile ilgili dizine geliriz.

csc deneme

derlenir.

.exe uzantılı dosya ekrana gelir ve Merhaba Dünya yazdırır.

**Visual Studio Ortam Tanıtımı**

-Bir platform'dur. Çok yönlü bir editördür.

-Solution explorer: Üzerinde çalıştığımız tüm dosyalar burada görülür.

-Error list'de derleme hatalarımızı görürüz.

- Proje: İçerisinde amaca dair çözümler getirilen kodsal çalışmaların-operasyanların yapıldığı bir bütündür.

- Solution: İçerisinde bir veya birden fazla proje barındırabilen bir evrensel kümedir.

Bir banka uygulaması birden fazla alt projeden oluşabilir. Bu projelerin tek bir çatı altında toplanması solution'dur.

Create a new project diyip yeni bir proje oluşturabiliriz.

Solution'a bir proje daha eklemek için solution'a sağ tıklayıp add project diyebiliriz.

Solution'u derlersek altındaki her proje derlenir.

Ya da sadece istediğimiz projeyi derleyebiliriz.

Build - derleme yapar

Rebuild - önceki derlenmiş dosyaları siler yeniden üzerine derler.

Clean - Derlenmiş dosyaları siler.

Build'a tıkladık dosya derlendi. Çıktıyı görmek için projeye sağ tıkla - open folder in file directory -> bin -> debug -> net5.0 'de derlemenin çıktılarını görebiliriz.

Clean dersek bu dosyalar silinir.

Eğer ki bir projeyi başlangıç projesi yapmak istersek sağ tıklayıp 'Set as Startup Project' dememiz gerekir.

**Visual Studio Code Nedir?**

Daha hızlıdır. Zayıf sistemler için idealdir.

Explorer penceresinde proje dosyaları görülür.

View - Terminal'de kendine ait dahili komut satırı vardır.

**Dotnet CLI:** Bir komut satırı arayüzüdür. Kendi arayüzü yoktur, komut satırından talimatlarla çalışır. Bir proje oluştururken hangi dizinde oluşturduğunuza

dikkat edin! .NET Uygulamalrı geliştirmeyi, oluşturmayı, çalıştırmayı, yayınlamayı sağlar.

cd C:\Users\acdem\Masaüstü\C#VSCode ile ilgili dizine girdik.

dotnet --işlem ile ilgili SLI komutlarını çalıştırabiliriz.

dotnet new console --name OrnekProje ilgili dizine OrnekProje isimli bir konsol uygulaması oluştur.

(projectType) --name ProjectName

MVC-WebAPI

.NET SDK'sı ile gelir. Geliştirme kitidir.

Dotnet [command] prototipindedir.

**Temel CLI Komutları:**

- Help : Tüm komutların ayrıntılı bilgisini veren komuttur.

- New : dotnet new [project type] --name [project name]

new --force ile aynı isimli projenin üstüne 'ezerek' bir proje oluşturulabilir.

- Restore : Proje sürecinde referans edilen yahut referansı kaldırılan paketlerin restorasyonunu sağlar.

Önceden varolan ya da hazırlanmış paketleri, kütüphaneleri kendi uygulamamızda kullanmak için çekmemiz lazım.

Bu paketleri uygulamaya yükleyebilmek için restore etmemiz gerekir.

dotnet restore ile kullanılır. NuGet havuzu kullanılır.

- Build

Bir projeyi derleyebilmek için varsayılan olarak bir restore işlemine tabi tutulur.

Projeyi derler ve sonuç olarak exe dll çıktıları verir.

\bin\Debug\net5.0 dizinine çıktı verir.

- Publish

Projeyi derleyerek, yayınlanabilir hale getirir. Public hale getirmeye yarar.

\bin\Debug\net5.0\publish dizinine çıktı verir. Çıktı olarak;

.dll

.deps.json(projenin tüm bağımlılıklarını(dependency-bir uygulama başka bir uygulamayı-kütüphaneyi kullanıyorsa bu bağımlılıkları referanslayan bir dosya verecektir) içerir.

.runtimeconfig.json(runtime konfigürasyonları)(Uygulama bağımlılıklarının dll'leri)

dotnet publish komutu.

- Run

dotnet run : derler ve ayağa kaldırır. Çalıştırır.

dotnet run --no-build : derlemeden ayağa kaldırır.

**Proje Modifikasyon Komutları;**

reference: önceden yazılmış .dll'lerdir.

NuGet havuzundaki .dll'lere ise paket-package denir.

bir yazılımı başka bir yazılım kullanacaksa çıktısı .dll

kullanıcı kullanacaksa çıktısı .exe'dir

add package : dotnet add package ilgili kütüphane ismi

add referance : dotnet add [source project].csproj reference [target project].csproj

-uygulamaya fiizksel bir dll dosyasını referans etmemizi sağlar.

dotnet add myproject.csproj referance ../ornekproje/ornekproje.csproj

myproject de örnek projedeki dll'i referans olarak al. Onun dll'inin çıktısını burada kullan.

remove package : dotnet remove package [package name]

remove referance : dotnet remove reference [reference name]

list referance : dotnet list reference

Bir uygulamada referans edilen tüm paketleri listeler.

**Programlaya Başlarken Temel İlkeler**

- Don't Repeat Yourself

- Anlamlı İsimlendirme

**Main Fonksiyonu Nedir?**

- Tüm uygulamalarda olması zorunludur.

- Uygulamalarda Program.cs dosyası başlangıç kodlarının bulunduğu yani uygulamanın

ayağa kalkabilmesi için başlangıç kodlarının bulunduğu bir dosyadır.

- Uygulama ayağa kalktığında işletim sistemi ile iletişim kurabilecek metodun ve bu metod içerisinde

başlangıca dair komutları barındıracak bir inşa = başlangıç kodları.

- Uygulamayı çalıştırdığımızda bu uygulamanın bulunduğu işletim sistemi ile iletişimini

sağlayan fonksiyona main fonksiyonu denir.

- Başlangıç kodları main'in içine yazılır.

- Herhangi bir uygulama olsa da bu main fonksiyonundan sade ve sadece 1 tane olmalıdır.

- static void Main (string [] args) {.....}

- Bu fonksiyon uygulama ilk ayağa kalktığında ilk tetiklenen fonksiyondur. Dolayısıyla

ilk kodlarımızı bu fonksiyonda yazarız.

**Top Level Statements Özelliği**

- Sıradan bir işlem için bile oluşturulan console uygulamasında bile basmakalıp

(boilerplate) kodların gelmesi gerekmektedir.

- Günlük hayatta basit kodlar inşa edebilmek ve test süreçlerinde hızlı denemeler

yapabilmek için bir kod bloğuna maruz kalmak gerekli değildir.

- C# 9.0 ile gelen top level statemensts özelliği ile main fonk. zoraki imzasının tanımlanması

kaldırılmıştır.

- Main fonksiyonunun kullanılması developer'in kararına bağlıdır.

- Normalde C# 9.0'dan önce uzun uzun yazmak gerekirken,

using System;

namespace ProjectExample

{

internal class Program

{

static void Main(string[] args)

{

Console.WriteLine("Hello World!");

}

}

}

Şimdi yerine sadece Top Statements Özelliği ile;

using System;

Console.WriteLine("Hello World!");

yazmamız yeterli. Bu şekilde yazılan komutlar derlendikten sonra esasında bir Main

fonksiyonu içerisine alınacaktır.

Uygulama derlenirken Program.cs dosyasında varsa Top Level Statements özelliği

bu dosyaya özel algılayacak ve ilgili alana yazılan kodları main içerisinde yorumlayacaktır.

Bunun dışında zaten bu özelliği başka bir dosyada kullanamyacağımızdan dolayı sadece

Program.cs dosyasına has bir özelliktir.

**Kurallar**:

1- Using blokları ile namespace arasında kodlar yazılabilir.

2- Bu işlem sadece Program.cs dosyasında geçerlidir. Main fonksiyonunda yazılacak komutların

direkt burada yazılmasında müsaade edilmekte lakin farklı bir dosyada bu işlemi gerçekleştirememekteyiz.

Genellikle bu özellik microservices yapılanmasında kodun gelişimi açısından hız kazandırıcı

bir niteliğe sahiptir.

**Not:** Mikroservis sadece bir işi yapan, bir fonksiyonaliteyi gerçekleştiren çok küçük kod parçacıkları.

**//** (Studio'da) ile yorum satırı eklenebilir.

// Açıklama satırları / yorum satırları

// Kodun niteliğini, anlaşılabilirliğini, kalitesini arttırmak için kullanılırlar.

// Kritik noktalarda ve özet olarak kodları izah etmeliyiz.

// Tek satırlık

**/\***

Çok

satırlık

yorum

alanı

**\*/**

// Nerede kullanılırlar? : İstenilen her yerde, semantik(anlamsal) akışı bozmaman kaydıyla…

**Region :** Kod dosyasını kategorik hale getirmemizi sağlayan bir ön işlemci(editör yorumlar) komutudur. Developer'ın yazmış olduğu kodu daha net görmesini sağlar.

#region A operasyonu

Binlerce satır kod...

#endregion

-'ye bastığımızda kategorize eder. Alt başlıklar olarak kullanılabilir.

**Todo Nedir?**

//todo Burada 1'den 10'a kadar yazılmalıdır.Yorum satırının içine yazılır.

View -> Task List'e tıkladığımızda görebiliriz.

Yorum satırının task olarak adlandırıldığı komuttur.

**Debugging Nedir?**

- İçerisinde bol bol mantıksal bir çalışma yapılan bir kod inşa ettiğimizi düşünelim.

- Debugging hata ayıklama demektir. Programın hatalarını yok etmeye yönelik yazılan kodu

gözden geçirme, düzeltme aktiviteleridir.

- Kodun akışı esnasında kodu okuyabilmemizi ve gözlemleyebilmemizi sağlar.

- Kompleks uygulamalarda zaman alıcıdır.

Takip etmek istenilen kodun satırının tam sol karşısına işaret konulur. Bu yuvarlak kırmızı işaretin adı breakpoint'dir. Kodun debug edilmesini sağlar.

Uygulamamız debug modda ayağa kaldırıldığında akış, breakpoint'in işaretlediği koda geldiğinde biz kodu debug edebileceğiz.

Kodun devam edebilmesi için F10 tuşuna basmalıyız. Eğer ki F5'e basarsak debug'u sonlandırmış oluruz.

**Watch Penceresi**

- Debug esnasında değişkenlerin değişen değerlerini görmekte kullanılır.

- İlgili değişkene sağ tıklayıp 'add watch' diyelim. Watch penceresinden akışa göre

değişkenlerin son durumunu görebiliriz.

- Debugsız Uygulama Çalıştırma : Zaman maliyetini önlemek için debugsız çalıştırılabilir.

Start Without Debugging ya da CTRL + F5 ile debug etmeden çalıştırırız. Daha hızlı çalışır.

**DERS - 2 :DEĞİŞKENLER**

- İşlenecek veriler RAM'de tutulur. Yazılımda veri tutulmaz. RAM'den alınıp RAM'e verilir.

Bunun için değişkenler kullanılır.

- Yazılımda işlenecek veriyi yazılım adına RAM'e yerleştirebilmek için değişkenleri

kullanırız.

- C# Programlama dilinde RAM'de veri tutabilmek/depolayabilmek için tanımlanacak olan

değişkenin türü/veri türü bildirilmelidir. Tür elimizdeki veriye göre bildirilir.

- Bir değişkenle RAM'de alan tahsisinde bulunulduğunda buna Değer Türlü - Value Type

değişken diyoruz. Yani tuttuğu değer bir normal değer olan değişkenlerdir.

**Değer Türlü Değişkenler:** Sadece bir değeri tutan değişkenlerdir. Ad, soyad, doğum tarihi..

**Primitive Type :** En ilkel tür. byte: primitive türdür. Byte'lardan bir araya gelmiş olan

decimal ise primitive değildir lakin value type'dır.

Değişken tanımlarken RAM'e tutulacak veriye uygun bir alan tahsisinde bulunması

gerekecektir. RAM'de alan tahsisinde bulunabilmek için ilgili değişkenin türünde

hareket edilir.

Bir türde tanımlanmış alana farklı bir türde değer atayamayız.

bool - mantıksal - doğru ya da yanlış : 0-1

char - metinsel - karakterler

kalanı sayısal. byte, short, int, float, double...

- Değişkenlerin max-min aralıkları vardır. Min değere uyarak optimize tür belirlemek önemli olabilir.

- string özel durum. Bir değer türü değildir bir referans türüdür. İleride!

Console.WriteLine(typeof(tür).**IsPrimitive**) -> fonksiyonunu çağırarak primitive kontrolü yapılabilir.

decimal, int, byte -> False, true, true yanıtlarını alırız.

**C# Temel Kuralları**

{.....kod mod cort....} -> Scope, faaliyet alanı.

Dil Özellikleri :

1. Bu dil büyük küçük harf duyarlılığına sahiptir. Ahmet ile ahmet farklıdır.

2. Tip güvenliği olan bir dildir.

**Değişken Tanımlama**

**Prototip** :

degisken\_turu degisken adi;

; ile kod konseptini kapatırız.

string adi; //RAM'de adi isminde string türünde bir değişken.

int yasi; //ayni şekilde

bool medenihal;

string soyadi;

RAM -> (int) yasi

(bool) medenihal

(string) soyadi

olarak oluştu.

**RAM'in Yapısı :** 2 kısımdan oluşur.

**Stack:** İçerisinde değer türlü değişken ve değerlerini tutabildiğimiz bölümdür.

Değişken türü, adı ve değerini tutar. Metot isimlerini de tutar.

İleride göreceğimiz metodların bellek adresleri STACK'de tutulur.

Not: Burada tutulan değişken ismine nadirde olsa referans dendiğini görebilirsiniz.(Değer türlü olsa bile, karıştırılmaması lazım. Lafın gelişi bir deyimdir.)

**Heap:** Nesneleri tutabildiğimiz bölümdür.

Yazılan değişkenlere koda yazılır yazılmaz alan tahsisinde bulunulmaz. Kod ne zaman çalıştırılır,

o alan compiler tarafından derlendikten sonra kodlara karşılık alan tahsis edilir.

STACK, FIFO(First in First out) denilen mantığa göre çalışır. İlk giren ilk çıkar.

Kodda yukarıdan aşağıya senkronizasyonda RAM'e yazılma aynı sırada yapılır.

**Değişken Tanımlama Kuralları**

- Tutacağımız değerin anlamını belirtecek şekilde isimlendirme yapılmalıdır.

- Değişken isimleri ?/',. gibi özel karakterler barındıramaz. \_ ise istisnadır.

double stok\_adedi; -> doğru

- Değişken isimleri sayısal ifade ile başlayamaz. Ama sayılar kullanılabilir.

int sayi1; -> doğru

int 1sayi; -> yanlış

- Bir scope içerisinde aynı isimde birden fazla değişken tanımlanamaz.

**İsimlendirme Kuralları**

1- PascalCase : Her kelimenin ilk harfi büyük yazılmalı. AdSoyad, TcKimlikNo, Satislar

Kısaltma iki harfliyse iki harfte büyük. In/Out = IO ya da InOutStream = IOStream

2- Camel Case : İlk kelime haricindeki tüm kelimelerin baş harfi büyüktür.

satisDurumu, personelAdi, userName, user, userNamePassword

3- Snake Case : Tüm kelimeler küçük ve arları alt tire.

kullanici\_adi, isim\_soyisim

**Değişken isimlerini @ operatörüyle tanımlama :**

- Değişken isimlerinde programatik keyword kullanılamaz. Eğer ki bir değişken isminde programatik

olarak kullanılan bir keyword'ü vermek istiyorsanız bunu @ operatörü ile ezebilir ve verebilirsiniz.

string static; -> olmaz. static bir keyword.

string @static; -> olur. Artık normal bir metin.

Anlamlı keyword olmasa da olur. string x; ile string @x; aynıdır.

**Tanımlanmış Değişkene Değer Atama**

int x = 123;

= operatörü, assign - atama operatörüdür. Sağdaki değeri soldaki değişkene, field'a, property'e atar.

string adi = "Faruk";

string soyadi = "Darmad";

char basHarf = 'g';

Stack'de yukarıdan aşağıya sırasıyla mevcut hale geldi.

Tanımladıktan sonra da değer atayabiliriz.

int x; //stack de x oluştu.

.

.

.

x=120; //x'e 120 atandı.

Eğer ki bir deişken ismi assign operatörünün soluna çağırılıyorsa o alana değişkenin

kendisi gelecektir.

**Örnek**;

int a = 5; //Stack (int) a = 5

int b = 15; //Stack (int) b = 15

string c; //Stack (string) c

c = "Hilmi"; //Stack (string) c = "Hilmi" //Hilmi değeri 3. satırda atanacaktır.

**Örnek**;

int x = 5; //Stack (int) x = 5

char y = 'a'; //Stack (char) y = 'a'

bool p; //Stack (bool) p

delimal d = 3.14m; //Stack (decimal) d = 3.14

p = false; //Stack (bool) p = false 3. satırda üzerinde değişiklik yapılacaktır.

Bir değişkene atanan en son değer geçerlidir.

int a = 5; stack'de a = 5 sonra 15 sonra 20 oldu.

a = 15;

a = 20;

Tanımlanmış olan değişkene türüne uygun değer atanmalıdır.

**Değişkene Değer Atama Kuralları**

**- Metinsel Değerler**

**string:** metinsel ifadeler çift tırnak içerisinde yazılmalıdır.

"Ali", "1245789987" -> Bir sayısal olarak metinsel olarak turuluyorsa eğer yazılım

açısından o metinsel bir ifadedir. Üzerinde matemetik işlem yapılamayan "Ali"'den farkı

olmayan bir değerdir.

"Ali 28 yaşındadır." -> string

"" içerisine yazılan tüm değerler string keyword'ü ile karşılanmalıdır.

**- Karaktersel Değerler**

char: '' tek tırnak içerisine yazılmalıdır.

char x = 'g';

'.' , ',' , '?'

'g' -> karakterdir.

"g" -> string'dir.

**- Mantıksal İfadeler**

**bool:** true ya da false ile belirtilir.

bool medeniHal = false;

bool medeniHal = true;

- **Sayısal Değerler**

Sayısal ifadelere değer atarken direkt olarak değeri göndeririz.

Sayısal ifadelerde bir değer default olarak integer kabul edilir.(Önemli)

Ondalıklı sayılar, tüm ondalıklı sayılar tam sayıları karşılayabilir, kapsar.

**float** x = 3.14f ya da 3.14F

float türünde bir küsüratlı değer tutarken ilgili değerin sonuna f ya da F gelmelidir.

Benzer kurallar double ve decimal içinde geçerlidir.

**double** x = 3.14d ya da 3.14D ya da 3.14

Ondalıklı türlerde bir değer defaul olarak double'dır. O yüzden yanına d ya da D yazmasakta olur.

**decimal** x = 3.14m ya da 3.14M

Bir sayısal tür değer aralığına girmeyen değeri tutamaz.

**Tuple Türüyle Değer Atama**

- Tek bir syntax üzerinde birden fazla değişken tanımlamamızı sağlayan bir nesnedir.

(int a, int b, type c, type d ...)z; -> z tuple değişkendir. İçerisinde birden fazla

farklı veya benzer türde değişken tanımı mevcuttur.

(int a, string b)c = (5,"ahmet"); -> 5 a'ya ahmet ise b'ye gidecektir.

c.a -> int değişkeni getirecektir.

c.b -> string değişkeni getirecektir.

(int yas, string adi, bool medeniHal)kisi = (28, "Ali", true);

kisi.yas = 5; -> değeri değiştirmek istersek.

(int x, char y)z ;

z = (5, 'a');

**Literal Düzenlemeler C# 7.0**

- Kompleks sayısal ifadeleri \_ alt tire ile düzenleyebiliriz.

int sayi = 1000000; yerine int sayi = 1\_000\_000; yazabiliriz.

**Değişkenlerin default değerleri;**

İleride OOP denilen yaklaşımı gördüğümüzde class içerisinde tanımlanan değerlerin

default değerlerinin otomatik atandığını konuşacağız.

string : null (değersiz), boş demek değildir.

char : '0'

sayisal ifadelerde : 0

bool : false

Default keyword'ü içerisine verilen türün varsayılan değerini geri döndürür.

bool x = default(bool); -> x false

int y = default(int); -> y 0

string z = default(string); -> z null

char c = default(char); -> c \0

Main içerisinde oluşturulan değişkenlerin ilk değerlerini manuel atmaya özen gösterilmelidir.

**Literal Düzenlemeler C# 7.1**

Üstteki default'lara değişiklik geldi.

bool x = default; ->dememiz artık yeterli.

**Tanımlanmış Değişkenin Değerini Okuma**

int x = 5;

- Bir değişkenin değerini elde edebilmek için değişkenin isminden faydalanılırız.

Bir değişkenin adı assign = operatörünün sağında yahut metodların parametrelerinde

çağırılıyorsa ilgili değişkenin değeri gönderilir.

Console.WriteLine(x); //x'in kendisi değil değeri gidecektir.

int y = x; //x'in değerini y'ye gönderir. Gelen değer y'ye atanır.

STACK

(int) x=5

(int) y=5

**Örnek**;

STACK

int a = 5; (int) a = 5

int b = 10; (int) b = 10 -> b = 5

int c = b; (int) c = 10 -> c = 5

int d = a; (int) d = 5

b = a;

c = b;

**Örnek**;

STACK

int a = 5; (int) a=5 -> a=5 //a'nın önceki varolan 5 değeri silinecek, 5 değeri tekrardan a'ya verilmiş olacak.

a = a;

**Değeri Olmayan Değişkenler**

Class içerisinde tanımlanan değişkenlerde varsayılan değer otomatik atanır.

Main içerisinde atanma olmaz. İlk değeri manuel vermediğimiz sürece işlem yapamayız.

Main gibi bir metod içerisinde tanımlanan değişkenlerin ilk değerlerini manuel olarak

vermeyi alışkanlık haline getirmek gereklidir.

**Değişken Davranışları Genel Bakış**

Örnek;

int x = 5;

int y = x;

y = 10; //x=5 ve y=10. Çünkü x sadece değerini gönderdi.

**Scope Kavramı :** Faaliyet alanı, kapsam. Değişken ve fonksiyonların erişilebilirlik sınırlarını

belirleyen alandır. Tanımlamalarda ve algoritmik çalışmalarda karışıklığı önleyen

bir sınır çizer.

C#'da süslü {} parantez ile.

Örnek;

{

int a = 5;

ConsoleWriteLine(a);

//Buradan erişilebilir.

{

//Buradan da erişilebilir.

ConsoleWriteLine(a);

}

}

{

ConsoleWriteLine(a); -> Çalışmaz.

//Fakat buradan a'ya erişilemez. Farklı bir scope'a geçtik.

}

Bir scope içerisinde tanımlanan o scope içinde her yerde erişilebilir.(Tanımlandıktan sonra)

- Bir scope'da aynı isimden birden fazla değişken olamaz.

- Farklı scope'larda aynı isimde değişkenler olabilir.

- Bir değişken sadece tanımlandığı scope içerisinden erişilebilir ve kullanılabilir.

**Değişmezler/Sabitler(const)**

- Sabitler, değişmeyen değerleri tutmak için oluşturulmuş yapılardır.

- Süreçte var olan değeri değiştirilemez, değiştirilmeye çalışıldığı taktirde compiler

tarafından hata verilir.

**const :** constant, değişmeyendir.

- Prototip olarak değişkenlere çok benzer lakin davranışsal olarak değeri bir daha değiştirilemez.

- Özünde static bir yapılanmadır. (Static, uygulama bazlı veri depolayabildiğimiz bellekte bir alandır. O alana veri koyabilmek için static olarak işaretlemek lazım.)

Static'e konulan bir değer evrenseldir. Uygulamadan her yerden erişilebilir.

- Static ile const arasındaki fark; static değişkenler adı üzerinde değişkendir, lakin const'lar sabittir.

- Tanımlanır tanımlanmaz değerlerini almak zorundadırlar.

**readonly:** Sadece okunabilir değişkenler tanımlamaktır.

- Const'tan farkı sadece tanımlandığı yerde değil, ayrıca constructor içerisinde de değeri

atanabilir. Dependency Injection deseninde çok sık tercih edilir. İleride detaylandırılacak!

- Readonly, static değildir.

double pi = 3.14; -> Sabit yapsak daha iyi!

const değişken\_tipi değişken\_tipi;

Bir const tanımlandığında STACK'de ilgili türde alan tahsis edilecektir ve ilk atanan

değer dışında bir daha değer kabul görmeyecektir.

Const'lar değiştirilemez lakin istenildiği kadar okunabilir, değerleri elde edilebilir.

const double pi = 3.14; //Tanımlama aşamasında değerlerini isterler.

pi = 123; -> Hata verir.

**Global Değişkenler**

- Bir değişken class scope'u içerisinde tanımlanıyorsa buna global değişken diyoruz.

class Program

{

int a; // Global değişken.

static void Main(...){

}

}

**Değişken Tanımlana Varyasyonları**

1) int a = 5;

int b = 10;

2) Aynı türden birden fazla değişken oluşturulacaksa eğer tek imzada tanımalayabiliriz.

int a = 5, b = 10 ;

int a, b;

**Değişkenler Arası Değer Atama Durumları - DEEP COPY - SHALLOW COPY**

- Değişkenler arası değer atanırken verisel açıdan iki davranış söz konusudur.

**Deep Copy:** Derin kopyalamada eldeki veri çoğalır/klonlanır. RAM'de bir veri varken aynı veriden bir tane daha olacaktır.

- Değer türü değişkenler birbirlerine atanırken default olarak deep cop geçerlidir. Yani veri otomatik olarak üretilir.

int a = 5;

int b = a;

a = a \* 5;

a -> 25

b -> 5 //a'da yapılan değişiklik b'ye yansımaz. Çünkü 5 değeri deep copy yapıldı.

**Shallow Copy:** Değişkenler arası değer atamalarında değeri türetmek/klonlamak/çoğaltmak yerine var olanı birden fazla referansla işaretlemeye dayalı kopyalama yöntemidir.

- Bellekte birden fazla referansın tek bir veriyi işaret etmesidir.

- STACK HEAP

A-------> | | Aynı veriyi işaret eder.

B-------> | |

- Neticede ilgili değer bir değişikliğe uğradığında tüm işaretleyen referanslara

bu değişiklik yansıyacaktır.

- Default olarak Shallow Copy referans türlü değişkenlerde geçerlidir.

İlerideki konusu olan ref keyword'ü ile Değer Türlü değişkenlerde nasıl Shallow Copy yapıldığını göreceğiz.

NOT: Shallow Copy OOP derslerinde ele alacağımız nesne ve referans arasındaki ilişkide varsayılan davranış olarak kabul edilmektedir.

**object Türü**

**object:** Tüm türleri karşılayabilen bir türdür.

- Tüm türler varsayılan olarak object'ten türerler. ((string, char, number, bool))

Bu yüzden tüm türleri karşılayabilmektedir.

- Referans türlü bir değişkendir. Lakin değer türlü değerleri de karşılayabilir.

string adi = "Ali";

object adi = "Ali";

- Object değişkenler ilgili değişkenleri RAM'de object türde

tutarlar. Lakin verinin öz türünü de içerisinde bozmadan saklarlar. Yani object içerisindeki

veri kendi öz türündeki veri olarak tutulur.

- Bu durum object içerisindeki veriyi kendi türünde tekrar elde edebiliriz anlamına gelmektedir.

Veriyi object içerisine koymaya -boxing- veriyi kendi türünde elde etmeye -unboxing- diyoruz.

**Boxing :** Object türdeki bir değişkene herhangi bir türdeki bir değeri göndermektir.

object yas = 28; //boxing. buraya yerleştirilen 28 int olarak yerleştirilir.

- Boxing işlemi neticesinde ilgili değer objectin içerisinde kendi türüyle saklanır.

- Lakin yaş değişkeni artık 28 değerini bize object türünde getirecektir.

- Burada object türden elde edilen değer üzerinden türüne özgü işkemler gerçekleştirilemez.

Örneğin sayısal bir değer varsa o değer object olarak geleceğinden matematiksel işlemler

yapılamaz.

- Object bir değişkenin içerisindeki değer üzerinde türüne özgü işlemler yapabilmek için

o objectin içerisinde değeri kendi/has/özgün türünde elde etmemiz gerekmektedir. İşte bu işleme de unboxing denir.

- Herhangi bir değer object türe assign ediliyorsa eğer bu işlem boxing'dir.

**CAST OPERATÖRÜ**

Boxing edilmiş bir veriyi kendi türünde elde etmemizi sağlayan bir operatördür.

- Tür dönüşümlerinde bilinçli tür dönüşümü konusunda ve kalıtımsal durumlarda cast operatörü kullanılır.

() cast operatörü parantezdir.

Cast operatörü object olan değişkenin solunda o object'i hangi türünde unboxing etmek istiyorsak parantez içerisinde

hedef türü bildirerek kullanılır.

int a = 5;

object b = a;

(int) b; // Cast operatörü b değişkeni/objesi içerisindeki değeri bana int olarak ver demektedir.

Örnek;

object \_yas = 28;

Console.WriteLine(\_yas \* 5); -> Hata.

int yas = (int)\_yas; //unboxing

Console.WriteLine(\_yas \* 5); -> Şimdi çalışır.

- UnBoxing esnasında Boxing edilmiş verinin orjinal türü neyse o bildirilmelidir.

- Yani int türünde boxing edilmiş bir değeri int türünde UnBoxing etmelisiniz. Chat türünde UB etmeye çalışırsanız! Patlarsınız.! Runtime error.

**var Keyword'ü;**

Tutulacak değerin türüne uygun bir değişken tnaımlayabilmek için kullanılan keyword'dür.

var keyword'ü kendisine atanan değerin türüne bürünür. (compiler'a bırakılır o iş)

var medeniHal = true; -> bool olduğundan dolayı var bool turune bürünecektir.

Bir keyword'dür bir tür değildir.

Esas kullanım amacı farklı diller arasında desteklenmeyen türlerdeki verileri karşılayabilmek için oluşturulmuştur.

Diller arasındaki entegrasyonda kullanılıyordu.

var yas = 25; -> Türünü bilebildiğimizverilerin dğeişken türlerini var ile compiler'a yaptırmak ufakta olsa maliyettir.

Bu maliyetten kurtulmak için manuel belirtmek daha iyi.

string adi = "Ali";

- var keyword'ü ile tanımlanan değişkenin değeri tanımlanma aşamasında verilmelidir. Ki türü belirleyip direkt ona dönüşebilsin.

- var keyword'ü ile tanımlanan değişkene ilk değer verildikten sonra o değerin türüne bürüneceği için sonraki

durumlarda değeri farklı türlerde verilemez.

var x = 3.14; -> var artık bir double'dır.

x = "Ali"; -> Hata verir. String değil.

- var ile object arasındaki fark; var bir keyword iken object ise bir türdür. var atanan değerin türüne bürünürken, object atanan değeri

boxing yaparak object'e dönüştürür.

**dynamic Keyword'ü**

var a = 5; -> compiler süresinde/development aşamasında a, int davranışı sergileyecektir.

dynamic a = 5; -> development aşamasında a'nın hala dynamic olduğu gözlemlenecektir. Ne zamanki uygulama derlenip çalıştırılır. O zaman dynamic ilgili değerin türüne bürünmüş olacaktır.

var derleme aşamasında değerin türüne bürünürken, dynamic ise runtime'da verilen değerin türüne bürünecektir.

Not: GetType() türün ne olduğunu söyleyen fonksiyondur. İşe yarayabilir.

Örnek;

dynamic a = "5";

Console.WriteLine(a\*5);

Kod compiler aşamasında hata vermez. Çünkü dynamic henüz atanmadı. Runtime'da hatayı verecektir.

Runtime'da a'nın string 5 değeri olduğunu anlar.

- dynamic keyword'ü runtime'da türü belirleyecektir. Lakin kararlı davranmayacaktır.

Örnek;

dynamic x = "Ahmet";

Console.Writeline(x.GetType());

x = 3.14D;

Console.Writeline(x.GetType());

Bu kod çalışır.Çalışma ekranında sonuç: System.String

System.Double olacaktır.

- Dynamic runtime'da esnek çalışmayı sağlar.

- Türünü bilmediğimiz bir veri geldiğinde runtime'da dynamic'te karşılayabiliriz.

- Uzaktan gelen verilen var keyword'ü ile karşılanamaz. Çünkü var keyword'ü tanımlandığı esnada verinin atanmasını ister.

**DERS-3: TÜR DÖNÜŞÜMLERİ**

**Kod Nasıl Çalışır?**

Yazılımlar varsayılan olarak senkron çalışırlar.

int a;

int b = 5;

string c = "Ahmet"; //Aşağı doğru tek tek ilerler.

Lakin ileri düzey prgramlama ile kodumuzu asenkron hale getirebilmekteyiz.

Genel olarak sağ'da işlem yap, sola assign et mantığını kullanırız.

**; noktalı virgül operatörü:** C# programlama dilinde kodun konseptinin bittiğini ifade eder.

- Kullanılmadığında compiler hata verir. Derlemez.

- Bir konseptin sonunda istenildiği kadar kullanılabilir.

int a = 5;;;;;;;;;; // Sorun yok. Ama çirkin.

int a = 123.ToString().Split().Rank; //konsepti sonlandırdık.

**NOT:** Kod aslında satır satır okumaz! Yukarıdan aşağıya okur diye bir kural yoktur!

Compiler kodu konsept konsept okur!

Örnek;

{

int

a

=

5

;

string b = "Jojo";

} // Derlenir.

**Tür Dönüşümü (Type Conversion) Nedir? Neden Verilerin Türlerini Değiştirmek/Dönüştürmek İsteriz?**

- Yazılım sürecinde elimizdeki değerlerin türlerini değiştirebilmekteyiz.

- Tür dönüşümleri elimizdeki veriyi türe uygun işlemlere tabii tutmak için uygulanabilir.

- Farklı servislerden gelen değerleri uygun türlere dönüştürmek isteyebiliriz.

- Bir int değeri decimal'e dönüştürerek aralığını arttırabiliriz.

- Tür dönüşümlerinde amaç türü dönüştürmektir. Yani elimizdeki veriye uygun bir türe geçiş yapmaktır. Elimizdeki veriyi uygun olmayan bir türe

dönüştürmeye çalışırsak bu mümkün değildir. Hata verecektir.

- Eldeki türe uygun dönüşüm yapılmalıdır. Amaç veriyi değiştirmek değildir. Amaç elimizdeki veriyi karşılayabilecek farklı bir türe dönüştürmektir.

Örnek;

{string}"435" -> int ve decimal dönüşür. Ama bool dönüşmez.

{string}"Ali" -> int, dec, char... dönüşmez.

{string}"a" -> int dönüşmez ama char dönüşebilir.

{string}"true"-> bool'a dönüşebilir ama int'e dönüşmez.

{int} 123 -> string ve char(istisna) dönüşür, bool'a dönüşmez.

**Metinsel İfadelerin Diğer İfadelere Dönüştürülmesi - Parse Fonksiyonu**

- Parse metodu/fonksiyonu sadece string dataları hedef türe dönüştürürken kullanılır!

type y = type.Parse(stringIfade);

Örnek;

string x = "123";

short x2 = short.Parse(x);

Console.WriteLine(x2 \* 5);

ya da

Console.WriteLine(short.Parse(x) \* 5);

Örnek;

string a = "Ahmet";

int a2 = int.Parse(a); //syntax'da problem olmaz. Runtime'da error verir. String harf, int'e dönüşmez.

Örnek;

string medeniHal = "true"; //string medeniHal = "Evli"; deseydik hata verirdi. Bool'da evli diye bir şey yok.

bool medeniHal2 = bool.Parse(medeniHal);

Console.WriteLine(medeniHal2); //string "" içerisine "true" ya da "false" yazdığı sürece hata vermez. Ekrana true ya da false yazdırır.

**Metinsel İfadelerin Diğer İfadelere Dönüştürülmesi - Convert Fonksiyonları**

Örnek;

string x = "25";

int x2 = Convert.ToInt32(x);

- Parse'da ne anlatıldıysa burada da geçerli.

Örnek;

string x = "3,14"; //nokta ile yazarsak ekrana 314 yazar. Burada virgül kullanılmalı.

double d = Convert.ToDouble(x);

**Diğer İfadelerin Metinsel İfadelere Dönüştürülmesi - ToString Fonksiyonu**

Örnek;

int a = 25;

string a2 = Convert.ToString(a);

Örnek;

float f = 35;

string f2 = f.ToString();

- Tür dönüşümlerinde genelde diğer türlerden string'e bir dönüşüm olduğu için bunu kolaylaştırmak adına ToString fonksiyonu oluşturulmuştur.

**C# Sayısal İfadelerin Kendi Aralarında Tür Dönüşümü!**

- Bir sayısal değer kendi türünden daha büyük değer aralığına sahip olan diğer türlere dönüştürülürken burada herhangi bir işlem yapmamıza gerek kalmayacağı için

bu işleme -bilinçsiz tür dönüşümü- denir.

- Bir sayısal değer kendi türünden daha küçük değer aralığına sahip olan diğer türlere dönüştürülürken hedef türün ilgili veriyi karşılayamama riskinden dolayı

buradaki işlemi bilinçli yapmamız gerekecektir -bilinçli tür dönüşümü-.

- Bir sayısal türün alt türüne bir veriyi dönüştürdüğümüzde eğer ki o veri alt türün değer aralığına girmiyorsa veri kaybı söz konusu olacaktır.

**C# Bilinçsiz Tür Dönüşümü - Implicit Type Conversion**

int a = 3000;

float f = a; -> şu anda burada bir tür dönüşümü söz konusudur. Lakin buradaki tür dönüşümü bizim kararımız/bilincimizle yaptığımız bir dönüşüm değildir.

Örnek;

short x = 123;

long y = x;

**C# Bilinçli Tür Dönüşümü - Explicit Type Conversion**

int x = 60000;

short y = x; //Hata.

- Boxing işlemlerinden tanıştığımız cast operatörü, bilinçli tür dönüşümünde de sayısal türleri kendi aralarında dönüştürürken iradeli bir şekilde bu işlerin

yapılmasını sağlayan bir operatördür.

int x = 3000;

short y = (short)x; //varsa bir veri kaybı ben (type) ile bunu bilinçli olarak üstleniyorum.

Değer aralığının modunu alıp sonucu ekrana yazdırır.

**Örnek; (Önemli)**

int a = 3000;

short s = (byte)a; //int'ten byte türüne bilinçli bir dönüşüm yapılmıştır. Sonra da byte'dan short'a atanarak bilinçsiz tür dönüşümü yapılmıştır.

**C# checked Bloğu İle Bilinçli Tür Dönüşümü Kontrolü**

checked {

//veri kaybı söz konusu olan kodları bu scope'ların içine yazmalıyız.

int a = 500;

byte b = (byte)a;

Console.WriteLine(b);

}

- checked, bilinçli tür dönüşümü esnasında bir veri kaybı söz konusu olursa eğer runtime'da bizleri uyaracak olan bir kontrol mekanizmasıdır.

**C# unchecked Bloğu İle Bilinçli Tür Dönüşümü Kontrolü**

unchecked{

int a = 500;

byte b = (byte)a;

Console.WriteLine(b);

}

- Hata vermez. Veri kaybı olur.

- Normal bir kod bloğu default olarak unchecked'dir.

**bool Türünün Sayısal Türe Dönüştürülmesi**

bool b = true;

int i = Convert.ToInt32(b);

// long i = Convert.ToInt64(b);

// short i = Convert.ToInt16(b);

// decimal i = Convert.ToDecimal(b);

Console.WriteLine(i);

- Ekrana 1 yazacaktır. False olsaydı 0 yazacaktı.

- Elimizdeki herhangi bir mantıksal değeri herhangi bir sayısal değere convert edersek ilgili değerin mantıksal karşılığını elde edebiliriz.

**Sayısal Türlerin bool Türüne Dönüştürülmesi**

int i = 1;

bool b = Convert.ToBoolean(i);

Console.WriteLine(b);

- Ekrana i = 0 ise False yazacaktır. 0 harici her değerinde True yazacaktır. (-123, 1000, 1,...)

- Tür dönüşümlerinde dönüştürülecek türün hedef türe uygun olması gerekiyordu. Burada bir istisna var.

**char Türünün Sayısal Türe Dönüştürülmesi (ASCII)**

- ASCII, bilgisayardaki her bir karakterin sayısal bir karşılığı vardır. Bu sayısal değerlere

ASCII kaynak kodu denmektedir.

- Örneğin, a harfi 97'ye A ise 65'e denk gelir.

char a = 'a';

int \_a = (int)a;

Console.WriteLine(\_a); // Ekrana 97'yi getirir.

- Cast operatörünü, unboxing, bilinçli tür dönüşümü ve burada da kullandık.

Sayısal Türlerin char Türüne Dönüştürülmesi

int oascii = 111;

int Oascii = 79;

Console.WriteLine((char)oascii); // ekrana küçük o

Console.WriteLine((char)Oascii); // ekrana büyük O yazdırılır.

**DERS-4: Programlamada Operatör Nedir?**

- Operatörler belirli bir sorumluluğu üstlenen sembolik yahut metinsel yapılardır. Bizim yerimize o sorumluluğu icra ederler.

- Operatörler genellikle iki değer arasında matematiksel, mantıksal ya da farklı bir işlemsel görev yapan yapılardır.

- Genellikle yaptıkları işlem neticesinde bir sonuç dönerler.

- Operatörleri kullanırken geriye dönüş değerlerine dikkat edilmesi gerekmektedir.

**C# Aritmetik Operatörler Neledir? Geriye Dönüş Değeri Nedir?**

- Aritmetik operatörler iki sayısal değer üzerinde işlem yapan operatörler oldukları için işlem neticesinde

geriye uygun türde sonuç dönerler.

- +,-,\*,/,%

Aynı türde olan sayısal değer üzerinde işlem yaparken sonuç türü aynı olacaktır.

- int sonuc = 3 + 5; // iki int değer int sonuc'ta tutulmalı.

- int x = 3, y = 5;

int sonuc = x \* y;

- double s1 = 123;

double s2 = 321;

double sonuc = s1 + s2;

- decimal s1 = 123123;

decimal s2 = 3214532;

decimal sonuc = s1 % s2;

İki farklı türde sayısal değer üzerinde yapılan aritmetik işlem neticesinde sonuç büyük olan türde dönecektir.

- int sayi1 = 10;

double sayi2 = 5;

double sonuc = s1 + s2; //sayi1'i double olarak hesaba katar.

**Not:** Aritmetik operatörlerde küçük olan tür büyük olan türe bilinçsiz bir şekilde dönüştürülerek o şekilde hesap yapılır.

o yüzden sonuç büyük olan türde elde edilecektir.

- int s1 = 3;

byte s2 = 123;

int sonuc = s1 - s2;

**- İSTİSNA!!! Önemli!**

byte s1 = 10;

byte s2 = 5;

int sonuc = s1 - s2;

Normalde 2 aynı türdeki sayısal değer üzerinde yapılan aritmetik işlem neticesinde sonuç aynı türde dönecekken, bu iki değer byte ise sonuç her zaman -int- dönecektir. Böyle kabul edilmiştir.

**C#'da Matematiksel İşlemlerde Öncelik Sırası Nasıldır?**

- Matematik prensipleri yazılımda da aynı şekilde geçerlidir. İlk olarak parantez işlemleri yapılır.

- Eğer soruda üslü ya da köklü bir sayı varsa ikinci adımı bu işlemlerin çözülmesi oluşturmaktadır.

- Sonrasında matematik işlemlerinden olan çarpma ve bölme yapılmalıdır.

- Dördüncü sırada toplama ve çıkartma işleminin yapılması gelmektedir.

- Son olarak da güdülmesi gereken çoklu işlemlerde işlem sürecinin soldan sağa doğru ilerlemesi olacaktır.

**C# Karşılaştırma Operatörleri Nelerdir?**

- İki sayısal değer arasında büyüklük, küçüklük ve eşitlik durumuna göre karşılaştırma yapan operatörlerdir.

- < küçüklük, > büyüklük , <= küçük veya eşitlik , >= büyük veya eşitlik , == eşitlik

- Karşılaştırma operatörleri geriye daima bool türünde bir değer döndürecektir.

Örnek;

int i1 = 123;

int i2 = 321;

-> i1 == i2, false

-> i1 < i2 , true

**C# Mantıksal Operatörler Nelerdir?**

- Tüm şartları değerlendirip, kendine göre sonuç döndüren operatörlerdir.

- ve && operatörü: Tüm şartları yerine getirilmiş olmasını ister.

- veya || operatörü: Şartlardan en az bir tanesinin yerine getirilmesi yeterlidir.

- ya da ^ operatörü: Şartlardan kesinlikle bir tanesinin yerine getirilmesini ister. Hiç ya da hepsi değil, sadece 1'i olmalı.

- Mantıksal operatörler, mantıksal değerler üzerinde kullanılır. Geriye mantıksal sonuçlar dönerler.

Örnek;

bool patates = true, kofte = false;

-> bool sonuc1 = patates && kofte; // False

-> bool sonuc2 = patates || kofte; // True

-> bool sonuc3 = patates ^ kofte; // True

**Mantıksal Operatörlerde Geriye Dönüş Değerleri**

- ve &&

true && true; -> True

true && false; -> False

false && false; -> False

false && true; -> False

- veya ||

true || true; -> True

true || false; -> True

false || false; -> False

false || true; -> True

- ya da ^ (shift + 3 + boşluk)

true ^ true; -> False

true ^ false; -> True

false ^ false; -> False

false ^ true; -> True

Örnek;

Console.WriteLine(((true && true) || false && ((true ^ false) && false) || true));

-----T------ -----T------

---------------T----------- -------------F-----------------

----------------------------F----------------------------------

------------------------------------------------------------------T-----------

Sonuç True olacaktır.

**C# Arttırma(++) Azaltma(--) Operatörleri**

++,-- operatörleri işlem neticesinde değişkenin değerini döner.

++i,i++,--i,i--

i=i+1;

i=i-1;

int i = 10;

++1 -> Öncelikle i değeri 1 arttırılır, ardından i değerini döndürür.

i++ -> Öncelikle i değerinni döndürür, sonra +1 arttırır.

Eğer ki ayrı ayrı;

CW(++i); -> Çalıştırırsak önce 1 arttırıp ekrana 11 yazar. Bellekte de 11 tutar.

CW(i++); -> Bunu çalıştırırsak önce yazdırır. Ekrana 10 yazar. Ama bellekte 11'i tutar.

Örnek;

int a = 5;

int b = a++; -> Önce a'nın değerini b'ye gönderir. Sonra a'yı 1 arttırır.

CW(a);

CW(b);

Çıktı: 6

5

olur.

Örnek;

int i1 = 5; -> (int) i1 = 5

int i2 = ++i1; -> i1'in değeri 1 arttı. i1 = 6, i2 = 6

int i3 = i1; -> i3 = 6

i2 = ++i2; -> Önce i2'nin değeri 1 arttır. i2 = 7, eski 7 ezilip yerine yeni 7 yazdırılır. i++ yazsaydı arttırmadan 6 yazardı.

CW(i1); -> 6

CW(i2); -> 7

CW(i3); -> 6

**Üzerine Ekleme/Yığma Operatörleri**

+=, -=, \*=, /=, %=

i += 3; -> i = i + 3; //sadece += metinsel ifadelerde de kullanılır.

diğerleri de aynı mantıkta. Pratik aritmetik işlemler yapmayı sağlarlar.

**Metinsel İfadelerde Kullanılan Operatörler**

- Metinsel ifadeler + operatörüyle yanyana birleştirebilirler.

string a = "Ahmet", b = "Mehmet";

CW(a+b); -> Çıktı AhmetMehmet olur.

Örnek;

int a = 5;

string b = "abc";

CW(a + b) ; -> Tür dönüşümlerinde herhangi bir değeri string'e dönüştürebilmek için ToString fonksiyonunu kullanıyorduk.

Ayrıyetten ilgili türü string'e dönüştürebilmek için string bir ifade ile + operatörüne tabi tutulması yeterli olacaktır.

Sonuç string dönecektir ve yanyana birleştirme olacaktır.

Çıktı: 5abc olacaktır.

Örnek;

+= operatörünün kullanılması.

string a = "ahmet";

string b = "mehmet";

a += b; -> a = a + b;

CW(a); -> çıktı ahmetmehmet olacaktır.

Örnek;

== operatörünün kullanılması.

string a = "Ahmet";

string b = "Mehmet";

bool sonuc = (a == b); // ya da a != b ile eşit değildir durumunu kontrol edebiliriz. O zaman da sonuç true olur.

CW(sonuc); -> False

**C# Operatörler - ! Operatörü**

! -> Değil, olumsuzluk.

Mantıksal yapılarda olumsuzluk ifade eder.

!true -> false

!false -> true

!= -> Eşit değillik durumu.

Ya da null referance durumunda kullanılır. (İleride)

Örnek;

CW(!true); -> False yazdırır.

Örnek;

int i = 3;

int i2 = 5;

CW(i != i2);

ya da

CW(!(i == i2)); -> ile kontrol edebiliriz. Parantez içi boolean bir değer döndürüyor.

! operatörü, !='in dışında sadece mantıksal değerlerin yanında kullanılabilir. !ahmet, !3 diyemeyiz.

**C# Operatörler - Ternary Operatörü (?:)**

- Şarta bağlı değer döndüren bir operatördür.

- Bir değişkene/metoda/property'e değer atanırken, eğer ki bu değer şarta göre fark edecekse satır bazlı/tek satırda bu şart kontrolünü yaparak

durua göre değeri döndürmemizi sağlayan bir kalıpsal operatördür.

- Ternary operatörü uzun bir if else bloğu yazmaktansa buradaki maliyeti tek satırda döndürmekte ve işartı kendi kalıbında kolayca kontrol edip

sonucu döndürmektedir.

........şart/durum......... ? .......... : ............

boolean bir sonuç döndürür ? şart true : şart false

ise bölgelerdeki değer döndürülecektir.

- Dönecek değerler aynı türde olmalıdır.

**Not:** Polimorfizm kurallarına göre birbirinden türeyen değerlerde artık desteklenmektedir. (ileride) (C# 9.0 de gelen bir özelliktir.)

Örnek;

bool medeniHal = true;

string mesaj = medeniHal == true ? "Evlilere kampanya...." : "Bekarlara kampanya...";

Örnek; // Birden fazla durumu kontrol etme.

// Yaşı 25'den büyük olanlara A, 25 olanlara B ve 25'den büyük olanlara C değerini döndüren ternary operatörü.

int yas = 25;

string sonuc = yas < 25 ? "A" : (yas == 25 ? "B" : "C"); -> 25'ten küçükse "A" döndür. Parantez şart değil. Yaş 25'e eşitse "B" değilse zaten ilk başta küçüktür olduğu durumlar burada elendiği için 25'ten büyükler için "C" döndürecektir.

Örnek;

Kullanıcı tarafından girilen sayının aşağıdaki önergelere göre hesabını gerçekleştiren kodu geliştiriniz.

// sayı < 3 -> sayı \* 5

// sayı > 3 && sayı < 9 -> sayı \* 3

// sayı >= 9 && sayi %2 == 0 -> sayı \* 10

// sayı %2 == 1 -> sayı

// hiçbiri değilse -> -1

Console.WriteLine("Lütfen bir sayı giriniz.");

int sayi = int.Parse(Console.ReadLine());

// ya da string sayi = Console.ReadLine(); // ReadLine() kullanıcının girdiği sayıyı string olarak yakalayan/getiren bir komuttur. Kullanıcıdan değer bekler.

//int sayi = int.Parse(sayi);

int sonuc = sayi < 3 ? sayi \* 5 :

(sayi > 3 && sayi < 9 ? sayi \* 3 :

(sayi >= 9 && sayi %2 == 0) ? sayi \* 10 :

(sayi %2 == 1 ? sayi : -1));

System.Console.WriteLine("Sonuç: " + sonuc); -> Girilen değere göre sonucu verecektir.

Örnek;

Hava durumu turan string değişkenin değerine göre aşağıdaki önergeleri uygulayan programı yazınız.

"Yağmurlu" ise "Şemsiye Almalısın."

"Güneşli" ise "Bol bol D vitamini alman dileğiyle..."

"Kapalı" ise "Yağmur yağabilir."

string havaDurumu = "Yağmurlu";

Console.WriteLine( havaDurumu == "Yağmurlu" ? "Şemsiye Almalısın." : (havaDurumu == "Güneşli" ? "Bol bol D vitamini alman dileğiyle..." : "Yağmur yağabilir."));

**C# Operatörler - Atama(Assign) Operatörü**

int a = 5; -> \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

değişkenin kendisi değişkenin değeri

İleride referans türü değişkenlerde atama operatörünün sorumluluğu değişip referans operatörü olduğunu göreceğiz.

**C# Operatörler - Member Access(Üye Erişim Operatörü)**

int i = 5;

i. -> yazdığımızda integer üzerinden erişebileceğimiz bir sürü member/eleman'lar görürüz.

Member, bir türün altındaki kodlardır, erişebildiğimiz/çağırabildiğimiz propertyler/metodlardır. (ileride)

i.ToString(); // ya da ToString(). ile de devam edilebilir.

- Member Access, elimizdeki bir değerin türüne uygun elemanlarını/fonksiyonlarını/metodlarını/propertylerini/field erişmemizi/çalıştırmamızı/çağırmamızı sağlayan bir operatördür. Kodun devamını getirir.

**C# Operatörler - Cast Operatörü ()**

Genellikle tür dönüşümlerinde kullanılır.

- Boxing -> Unboxing

object x = 123;

int x2 = (int)x;

- Bilinçli Tür Dönüşümü

int a = 5 ;

short b = (short)a;

- Char -> int veya int -> char (ASCII)

int ascii = 93;

char c = (char)ascii;

İleride polimorfizm durumunda base class referansıyla işaretlenen bir nesneyi kendi türünde de elde edebilmemizi de sağlar.

**C# Operatörler - sizeof Operatörü**

- Verilen türün bellekte kaç byte'lık yer kapladığını integer olarak geriye döndürür.

CW("int : " + sizeof(int)); -> 4

CW("long : " + sizeof(long)); -> 8

CW("decimal : " + sizeof(decimal)); ->16

**C# Operatörler - typeof Operatörü**

- Verilen türün/değerin type'ını/türünü getirir.

- O tür ile ilgili bilgileri edinmek için kullanılır.

- İleride (ileri düzey programlamada) reflection konusunda reflection'lara girmek için kullanacağız.

Type t = typeof(int); //int türüne ait tüm bilgiler burada t değişkeninde atanmıştır.

CW(t.name); -> Int32

CW(t.IsPrimitive); -> True

CW(t.IsClass); -> False

CW(t.IsValueType); -> True

**C# Operatörler - default Operatörü**

- Belirtilen türün default değerini döndüren operatördür.

- Default değerler her tür için yazılım tarafından tanımlanmış bir varsayılan değerdir.

- sayisal = 0, bool = false, string = null, char = \0, referans = null

CW(default(decimal)); ->0

CW(default(string)); ->// string ifade null'dur, değersizdir.

CW(default(Program)); ->//Program bir referans türüdür.

CW(default(short)); ->0

CW(default(byte)); ->0

int a = default; //default değer vermek için direkt default keyword'ünü verebiliriz.

int a2 = default (int); //bu şekilde de yazılabilir.

**C# Operatörler - is Operatörü**

- Boxing'e tabi tutulmuş bir değerin öz türünü öğrenebilmek/check edebilmek/kontrol edebilmek için kullanılan bir operatördür.

is operatörü denetleme neticesinde durumu bool yani true ya da false olarak döndürecektir.

object x = true; //boxing

CW(x is bool); ->True

CW(x is int); ->False

CW(x is Program); ->False

İleride if yapılanmasında çok tercih edilen bir operatör olacaktır.

Yine OOP yapılanmasında polimorfizm, is operatörüyle kalıtımsal durumlardaki nesnelerin türlerini de öğrenebileceğiz.

**C# Operatörler - is null Operatörü**

- Bir değişkenin değerinin null olup olmamasını kontrol eden ve sonuç olarak geriye bool türde bir değer döndüren operatördür.

String a = null;

CW(a is null);

- is null operatörünü sadece null olabilen türlerde kullanabiliriz.

Değer türlü değişkenler not nullable; referans türlü değerler nullable'dur.

int a = null; -> olmaz.

**C# Operatörler - is not null Operatörü**

- Elimizdeki değerin null olup olmamasıyla ilgilenmekte ve geriye bool sonuç döndürmektedir.

string a = "sdaasdsad";

CW(a is not null); -> True

- Sadece null alabilen türlerde kullanılabilir.

**C# Operatörler - as Operatörü**

x as Type

- Cast operatörünün UnBoxing işlemine alternatif olarak üretilmiş bir operatörüdür.

Cast ile object x = 123; //yapılan unboxing işleminde türüne uygun bir şekilde Cast edilmesi gerekiyor. Farklı türde hata verir.

(int)x;

- as operatörü ise object x = 123; //Türüne uygun bir şekilde as edilmesi zaruri değildir. Eğer ki tür uygunsa unboxing başarılı, değilse de hata vermeden null değer döndürecektir. Bu durumda programlamatik olarak yazılımın sonlanmadan akışta kontrol edilmesine müsade edecek, işleme devam edecektir.

Type y = x as Type;

- as operatörü tür uygun olmadığı takdirde geriye null döndüreceği için bu null'u karşılyabileen türlerle çalışmak isteyecektir.

Dğer türlü değişkenler ile kullananılamayacaktır. Referans türlerdeki değişkenlerle çalışabilir.

Örnek;

object x = "Ahmet;

string x2 = (string)x;

CW(x2); -> Ekrana Ahmet yazdırılır.

object x = "Ahmet;

string x2 = (int)x; -> String ifadeyi int ile cast etmeye çalışırsak,

CW(x2); -> Hata verir.

object x = "Ahmet;

string y = x as string;

CW(y); -> Ekrana Ahmet yazdırılır.

object x = "Ahmet;

string y = x as int; -> Sadece nullable türlerde çalışır.

CW(y); -> Hata verir.

object x = "Ahmet;

Program y = x as Program; -> Bir başka nullable olan referans türüyle unboxing yaparsak,

CW(y); -> Ekrana bir şey yazdırmaz çünkü null döndürür.

**C# Operatörler - Nullable Operatörü**

- C# Programlama dlinde değer türlü değişkenler normalde null değer alamazlar.(not nullable)

- Bir değer türlü değişkenin null değer alabilmesi için(yani nullable olabilmesi için) ? operatörünün kullanılması gerekmektedir.

int? a = null; //Artık değişken türlüden ziyade referans türlüymüş gibi hareket edecek ve null değerlerini karşılayabilecektir.

bool? b = null;

Prototip -> değişken\_türü? değişken\_adi

- Artık is null ve is not null kontrolleri de yapılabilir. as de kullanabiliriz.

Örnek;

object x = 123;

int? y = x as int?; //x'i integer çıkarıyorsak nullable dönebilmesi için integer'ın nullable haliyle çıkarmaya dikkat etmek lazım. Aynı şekilde ?y'e atarken de.

**C# Operatörler - Null Coalescing Operatörü**

- Elimizdeki değişkenin değerinin null olma durumuna istinaden farklı bir değeri göndermemizi sağlayan operatördür.

string a = null;

a ?? "Merhaba" -> a değişkeninin değeri null değilse a'nın değerini yazdır. Null ise "Merhaba" yazdır.

Örnek;

string a = "Ahmet";

Console.WriteLine(a ?? "Merhaba"); -> a null olmadığından dolayı Ahmet yazdırır.

string a = null;

Console.WriteLine(a ?? "Merhaba"); -> a null olduğundan dolayı Merhaba yazdırır.

string a = null;

Console.WriteLine(a ?? 3); // null Coalescing operatöründe her iki taraftaki değişken yahut değerler aynı değerde olmalıdır. Hata verir.

**C# Operatörler - Null Coalescing Assignment(C# 8.0)**

string x = null;

Console.WriteLine(x ??= "Merhaba");

//x'in değeri null ise Merhaba yazdır ve Merhaba değerini x'e ata.

//null değilse direkt olarak x'in değerini yazdır.

Örnek;

int? id = null;

id ??= 1; //id null ise 1 değerini ata, değilse değerini koru.

**Ders-5: Akış Kontrol Mekanizmaları Nelerdir Ne Amaçla Kullanılırlar?**

- Akış kontrol mekanizmaları, kodun akış sürecinde belirli şartlara göre farklı yönlendirmeleri yapmamızı ve farklı

algoritmaları/kodları/yapılandırmaları çalıştırmamızı sağlayan mekanizmalardır.

- Yazılım kodunun akış süreciden şarta göre yönlendirilmesini sağlamaktadır. Algoritmalarda ciddi manada kullanılan yapılardır.

Olmazsa olmazdır.

- Akış kontrol mekanizmalarında if-else veya switch yapılanmaları aynı işi farklı şekilde yapmamızı sağlayan

birbirlerinden farklı yapılanmalardır. İkisi arasında teknik olarak fark olsa da işleyiş/kullanım açısından bir fark yoktur.

**Switch Case Yapılanması**

- Kodun akışında belirli bir şarta göre yönlendirme yapmamızı(farklı algoritme çalıştırmamızı ya da farklı operasyon

gerçekleştirmemizi/tetiklememizi) sağlayan yapılanmadır.

- Bu yapılanma sadece bir değişkenin değerinin sadece eşitlik durumlarını kontrol ederken kullanılabilir.

switch (süreçte eşitlik durumu kontrol edilecek değer)

{

case value1 :

break;

// switch yapılanmasında verilen değerin eşitlik durumlarını kontrol edeceğimiz diğer değerleri case bloklarına bu şekilde yerleştiriyoruz.

// : case value belirtildikten sonra kalıpsal olarak burada kullanılmalıdır.

case value2 :

break;

//case ile break arasına yazılan kodlar ilgili case'in doğrulanması sürecinde işlenecek kodlardır.

.

.

.

default:

//hiçbir case eşitlik durumunu sağlayamıyorsa default break arasındaki kodlar çalıştırılacaktır. Zorunlu değildir.

break;

}

- Switch case yapılanmasında kontrol edilen değer ile eşitlik sağlayan case varsa diğer case'lere bakmaksızın switch bloğundan çıkılacaktır. Amaç eşitlik durumuna göre belirli bir kod grubunu tetiklemektir.

Örnek:

string adi = "Ahmet";

switch (adi) //kontrol edilen değerin türü neyse case bloklarında da aynı türde değerde kontrol edilmelidir.

//Switch bloğunda kontrol edilen değer bir değişken yahut sabit ya da statik bir değer olabilirke,

{

case "Mehmet": //case bloklarındaki değerler kesinlikle sabit/statik olmak zorundadır.

CW("Adı Mehmet.");

break;

case "Ayşe": //case bloklarındaki değerler değişkenlerden alınamaz.

CW("Adı Ayşe.");

break;

case "Hilmi": //case bloklarının sıralaması da önemli değildir.

CW("Adı Hilmi.");

break;

default: //şart değil varsa tetiklenir. Blok içerisinde herhangi bir yere de yazılabilir

CW("Hiçbiri değil.);

break;

} //Eşleşmeyi bulur bulmaz switch bloğundan çıkacaktır(compiler). Diğer case'lere bakmaz.

**Akış Kontrol Mekanizmaları - Switch Case'de When Şartı**

- Switch yapılanmasında sadece elimizdeki değerin eşitlik durumunu kontrol edebilmekteyiz. Bunun dışında bu kontrol esnasında farklı şartları da değerlendirmek

istersek when keyword'ünü kullanabiliriz.

switch(satisTutari)

{

case 100 when (\_\_\_şart\_\_\_): // satisTutari hem 100'e eşit mi hem de şart doğru mu diye kontrol ediyoruz.

break;

}

Örnek:

int satisTutari = 1000;

switch(satisTutari)

{

case 1000 when (3 == 5): // satisTutari 1000 ise VE 3 == 5 ise dediği için buraya girmez.

break;

case 1000 when (3 == 3): // 1000 VE 3 == 5 true olduğu için buraya girer.

break;

}

**Akış Kontrol Mekanizmaları - Switch Case'de goto Keyword'ü**

- switch case yapılanmasında sadece eşitlik durumunu inceleyebildiğimiz için mantıksal bir işlem gerçekleştiremeyiz.

Dolayısıyla bazen farklı değerlere eşit olma durumunda aynı operasynu/kodu/akışı kullanacağımız senaryolarla karşılaşabilmekteyiz.

switch(value)

{

case "a":

A işlemini uygula

break;

case "b":

B işlemini uygula

break;

case "c":

A işlemini uygula //aynı kodu yazdık! olmaz.

break;

.

.

}

- Farklı eşitliklerde aynı kodu çalıştıracaksak eğer kod tekrarına girmemek için goto keyword'ü ile "şu case'deki kodu çalıştır." diyebiliyoruz. Case'ler arasında zıplama yapabiliyoruz.

Örnek:

int i = 10;

switch(i)

{

case 5:

CW(i\*10);

break;

case 6:

CW(i/5);

break;

case 7:

goto case 5; // break'i kaldır. case'deki kodu çalıştır. goto varsa break'i kulanmayız.

case 10:

CW(i\*10); // kod tekrarını çözmek için case'deki gibi yapabiliriz.

break;

//goto keyword'nün kullanıldığı case'in eşleştirmesine bakmaksızın direkt kodunu çalıştıracaktır.

}

- Birden fazla aynı case'e yönlendirilen goto keyword'ünü kullanmak için;

.

.

case 7:

case 10:

goto case 5; //her ikisinden biri olduğu durumda case 5'e yönlendir. case 7 veya case 10'sa 5'e yönlendir.

}

**C# 8.0 Switch Expressions Nedir?**

- Tek satırlık işlemler için maliyet düşürücü ve kullanışlı semantiklerdir.

Örnek:

// ESKİ YÖNTEM

string isim = "";

int i = 10;

switch (i)

{

case 5:

isim = "Hilmi";

break;

case 7:

isim = "Ahmet";

break;

case 10:

isim = "Mahmut";

break;

}

// YENİ YÖNTEM

int i = 10;

string isim = i switch

{

5 => "Hilmi", // => ise

7 => "Ahmet",

10=> "Mahmut"

};

// Yeni yöntemle çok daha az maliyetli oldu.

**C# 8.0 Switch Expressions - when Şartı Uygulamak**

int i = 10;

string isim = i switch

{

5 when \_\_\_şart\_\_\_=> "Hilmi", // => ise

7 when 3 == 3 => "Ahmet", //bu şekilde de yapılabilir ya da

var x when x == 10 && x%2 == 1 => "Mahmut", //bir değişken tanımlayıp bir condition ile VE'leyerek kullanabiliriz. i'yi var yerine int ya da kendi türünü karşılayan bir değişkenle de karşılayabiliriz.

var x => "Hiçbiri" // hiçbir eşleştirmenin olmadığındaki default tanımlamasına karşılık gelecektir.

};

**C# 8.0 Switch Expressions - Tuple Patterns**

- Tuple patterns ise switch yapılanmasını Tuple nesnelerini kontrol edebilecek şekilde hem standart hem de yeni yapılanmayla bizlere sunmaktadır.

Örnek:

// ESKİ-YENİ YÖNTEM

int s1 = 10;

int s2 = 20;

string mesaj = "";

switch(s1,s2)

{

case(5,10):

mesaj = "5 ile 10 değerleri";

break;

case(10,20):

mesaj = "10 ile 20 değerleri";

break;

}

Console.WriteLine(mesaj));

}}}

//DAHA YENİ YÖNTEM

int s1 = 10;

int s2 = 20;

string mesaj = (s1,s2) switch

{

(5,10) => "5 ile 10 değerleri",

(10,20)=> "10 ile 20 değerleri" //sonuncuya virgüle gerek yok.

};

Console.WriteLine(mesaj)); }}}

**C# 8.0 Switch Expressions - Tuple Patterns when Şartı Uygulamak**

Örnek:

int s1 = 10;

int s2 = 20;

string mesaj = (s1,s2) switch //tuple'ın türü birden fazla değişkenin bir araya gelmesiyle oluştuğu için ortak bir türle karşılanmıyor.

{

(5,10) when (true) => "5 ile 10 değerleri", //ile yapabiliriz ya da

var x when x.s1 % == 1 || x.s2 == 10 => "s1'in 2 modu 1 olduğunda ve s2'nin değeri 10 olduğunda."

// var kullanmak daha iyi. tuple çünkü.

};

Console.WriteLine(mesaj));

}}}

**C# 8.0 Switch Expressions - Positional Patterns ve when Şartının Kullanımı**

- Positional Patterns ise Deconstruct özelliği olan nesneleri kıyaslamak yahut değersel karşılaştırmak için kullanılan bir gelişimdir. (ileride OOP'de)

- \_ => alt tire ise hiçbiri anlamına gelmektedir.

- Yine aynı şekilde ise'den önce ya da değişken ile kullanırız. (bunların hepsi ileride sakin!)

**C# 8.0 Switch Expressions - Property Patterns**

- Nesnenin propertylerine girerek belirli durumları hızlı bir şekilde kontrol etmemizi gerçekleştiren ve bunu farklı değerler için birden fazla kez tekrarlı bir

şekilde yapmamıza olanak sağlayan güzel bir gelişimdir. (bu da OOP'de daha detaylı.)

- when şartı da her zamanki gibi.

**Akış Kontrol Mekanizmaları - if Yapısı**

- Switch case elimizdeki bir değerin sadece eşitlik durumnu check eden,kıyaslayan bir akış kontrolüydü.

- if yapılanması ise elimizdeki bir değerin eşitlik durumu da dahil bütün karşılaştırmaları yapmamızı ve duruma göre

akışı yönlendirmemizi sağlayan bir yapılanmadır.

Prototip:

if (\_\_\_şart\_\_\_) //Karşılaştırma operatörleri ve mantıksal operatörlerin hepsi burada kullanılabilir.

{

//Eğer ki şart doğru ise true ise burası tetiklenir.

}

//değilse compiler if scope'undan çıkacak ve yoluna devam edecektir.

- Şart her zaman bool türde olmalıdır.

Örnek:

bool medeniHal = true;

if (medeniHal == true)

{

CW("Allah bir yastıkta kocatsın.");

}

// if yapılanması tek başına kullanılıyorsa sadece şarta bağlı çalışacak koda odaklanır.

Örnek:

// if yapılanmasında illa ki else kullanmak zorunda değiliz.

int i = 10;

if (i == 10)

{

Console.WriteLine("Merhaba");

}

Console.WriteLine("Dünya");

Çıktı: Merhaba

Dünya

if (i != 10)

{

Console.WriteLine("Merhaba");

}

Console.WriteLine("Dünya"); //burası her daim çalışacak bir koddur. if ile alakası yok.

Çıktı: Dünya

Örnek:

bool medeniHal = true;

if (medeniHal) //illa karşılaştırma operatörü kullanmak şart değil. Zaten bool.

// ya da if(!medeniHal) yazarak false'ı da kontrol edebiliriz.

{

CW("Allah bir yastıkta kocatsın.");

}

**Akış Kontrol Mekanizmaları - if else Yapısı**

- if else yapısı şarta göre bir kodun çalışıp, şartın olmadığı durumda bir başka kodun çalışmasını belirleyen bir yapılanmadır.

Şartın olumsuz/değil durumunda çalışacak kodu belirlemiş oluyoruz.

if(\_\_\_şart\_\_\_)

{

//şart true ise sadece if.

}

else

{

//şart false ise kesinlikle else bloğu tetiklenir.

}

//ikisi aynı anda tetiklenmez.

Örnek:

int i = 10;

if(i>5)

{

CW("i değeri 5'ten büyüktür."); //burası yazdırılır.

}

else

{

CW("i değeri 5'ten küçüktür."); //i = 3 olsaydı burası.

}

Örnek:

int i = 10;

if(i == 10)

{

CW("i değeri 10."); //i'nin 10 olduğu durum if'te.

}

else

{

CW("i değeri 10 değil."); //i'nin 10 olmadığı durum else'de.

}

//kodu ile

int i = 10;

if(i != 10)

{

CW("i değeri 10 değil."); //burada ise i'nin 10 olmadığı durum if'te.

}

else

{

CW("i değeri 10."); // 10 olduğu durum else'de.

}

//aynı anlama gelir.

Örnek: //her iki durumda da aynı kodu çalıştırmak ve kod tekrarına düşmemek istiyorsak...

- if ve else bloklarında aynı işlemleri yapacaksak eğer tekrarlı bir biçimde yapmamamız gerekmektedir. Her iki durumda da ortak çalıştırılacak olan komutları if else bloğunun dışına yazmamız yeterli olacaktır.

int i = 10;

if(i == 10)

{

CW("i değeri 10."); //i'nin 10 olduğu durum if'te.

//CW("Merhaba"); olmaaz.

}

else

{

CW("i değeri 10 değil."); //i'nin 10 olmadığı durum else'de.

//CW("Merhaba"); olmaaaz.

}

CW("Merhaba"); //böylece hangi bloğa girerse girsin burası çalışacaktır ve iki kere yazmamız gerekmeyecektir.

**Akış Kontrol Mekanizmaları - if Else if Yapısı**

- Birden fazla şartı kontrol etmemizi sağlayan bir yapılanmadır.

- Eğer hava yağmurluysa şunu yap değilse bunu yap -> if else

Eğer haca yapmurluysa şunu yap yok yağmurlu değil kapalıysa bunu yap, o da değilse.... yap -> if - else if

Prototip:

if(\_\_\_şart\_\_\_)

{

// şart doğruysa burası

}

else if(\_\_\_şart\_\_\_)

{

// yok eğer yukarıdaki şart doğru değilse buradaki şart doğru ise burası tetiklenir.

}

else if(\_\_\_şart\_\_\_)

{

// aynı şekilde.

}

.

.

.

// if yapılandırmalarından herhangi biri doğrulandıysa eğer diğer if’ler değerlendirilmeyecektir.

Örnek:

int sayi = 30;

if(sayi > 5 && sayi <= 10) //false

{

CW(sayi \* 5);

}

else if(sayi > 10 && sayi <= 20) //false

{

CW(sayi \* 10);

}

else if(sayi > 20 && sayi <= 30) //true. içeri girdi ve tetikledi.

{

CW(sayi \* 20);

}

Örnek:

int sayi = int.Parse(Console.ReadLine()); // Kullanıcıdan sayıyı 210 girer isek.

if (sayi > 100 && sayi <= 200)

{

Console.WriteLine("100 ile 200 arasında."); //false, buraya girmez.

}

else if (sayi > 200 && sayi <= 300) // burayı tetikler ve sonuncusuna bakmadan if bloğundan çıkar.

{

Console.WriteLine("200 ile 300 arasında.");

}

else if (sayi > 200 && sayi <= 400) // burasını kontrol bile etmez. Burası da çalışsın istiyorsak eğer else if mantığıyla çalışmamız hatalıdır.

{

Console.WriteLine("200 ile 400 arasında.");

}

// kodu aşağıdaki gibi düzenlersek.

int sayi = int.Parse(Console.ReadLine()); // Kullanıcıdan sayıyı 210 girer isek.

if (sayi > 100 && sayi <= 200)

{

Console.WriteLine("100 ile 200 arasında."); //false, buraya girmez.

}

if (sayi > 200 && sayi <= 300) // Hem burası,

{

Console.WriteLine("200 ile 300 arasında.");

}

if (sayi > 200 && sayi <= 400) // Hem de burası tetiklenir. Ve ekrana iki CW'de yazdırılır.

{

Console.WriteLine("200 ile 400 arasında.");

}

// her bir if bloğu başlı başına bağımsız bir bloktur.

Örnek:

int i = 100;

if(i < 100)

{

CW("100'den küçük.");

}

else if(i > 100)

{

CW("100'den küçük.");

}

else

{

CW("100'e eşit.");

}

**Akış Kontrol Mekanizmaları - Scope’suz If Yapısı**

- Tek bir satırlık/konseptlik(;'e kadar) işlem yapılacaksa scope kullanmadan da yazılabilir.

if (true)

{

CW(".........");

}

ya da

if (true)

CW(".........");

- Eğer ki birden fazla k onsept/işlem/operasyon barındıracaksa bunları kesinlikle scope içine almamız gerekli. Aksi taktirde scopesuz ilk işlemi if bloğu alacak diğerlerini almayacaktır.

ÖRNEK:

// klavyeden iki ürünün fiyatı girildiğinde toplam fiyat 200 TL'den fazla ise, 2. üründen %25 indirim yaparak ödenecek tutarı gösteren uygulama.

Console.Write("Lütfen birinci ürünün fiyatını giriniz : ");

//write yazar, yadığı yerde kalır; writeLine yazar bir satır aşağıya geçer.

int birinciUrunFiyati = int.Parse(Console.ReadLine());

Console.Write("Lütfen ikinci ürünün fiyatını giriniz : ");

//write yazar, yazdığı yerde kalır; writeLine yazar bir satır aşağıya geçer.

int ikinciUrunFiyati = int.Parse(Console.ReadLine());

// int toplam = birinciUrunFiyati + ikinciUrunFiyati; ya da değişken oluşturmaya gerek yok diyelim.

if (birinciUrunFiyati + ikinciUrunFiyati > 200)

{

//int toplamSonTutar = birinciUrunFiyati + (ikinciUrunFiyati \* 75 / 100); //değişkeni sonradan tekrardan kullanacaksan oluştur. Yoksa oluşturma.

//yüzde 25 indirim demek 75 çarp 100 böl demek.

//ya da direkt olarak yazdırabiliriz.

Console.WriteLine(birinciUrunFiyati + (ikinciUrunFiyati \* 75 / 100));

}

else

{

Console.WriteLine(birinciUrunFiyati + ikinciUrunFiyati);

}

ÖRNEK:

// Belirlenen kullanıcı adı ve şifre doğru girildiğinde "Giriş başarılı", yanlış girildiğinde "Girdiğimiz kullanıcı adı veya şifre hatalı." mesajı veren console

// uygulamasını yazalım.

Console.WriteLine("Lütfen kullanıcı adınızı giriniz.");

string kullaniciAdi = Console.ReadLine();

Console.WriteLine("Lütfen şifrenizi giriniz.");

string sifre = Console.ReadLine();

if(kullaniciAdi == "Ali" && sifre == "12345") //tek satır işlem olduğu için {}'ları kaldırabiliriz.

//{

Console.WriteLine("Giriş başarılı.");

//}

else

//{

Console.WriteLine("Girdiğiniz kullanıcı adı veya şifre hatalı.");

//}

ya da

if(!(kullaniciAdi == "Ali" && sifre == "12345")) // şarta ! koyarak ters yüz de edebiliriz.

Console.WriteLine("Girdiğiniz kullanıcı adı veya şifre hatalı.");

else

Console.WriteLine("Giriş başarılı.");

ya da tek satırda...

Console.WriteLine(kullaniciAdi == "Ali" && sifre == "12345" ? "Giriş başarılı." : "Girdiğiniz kullanıcı adı veya şifre hatalı.");

ÖRNEK:

// Kullanıcıdan alınan iki sayının ve yapılacak işlem türünün (toplama, çıkarma, çarpma, bölme) seçilmesiyle, sonucu hesaplayan

// programı yazalım.

Console.WriteLine("Lütfen 1. sayıyı giriniz.");

int sayi1 = int.Parse(Console.ReadLine());

Console.WriteLine("Lütfen 2. sayıyı giriniz.");

int sayi2 = int.Parse(Console.ReadLine());

Console.WriteLine("Lütfen yapılacak işlemi belirtiniz.(+,-,\*,/"));

char islem = char.Parse(Console.ReadLine());

if (islem == '+')

{

Console.WriteLine(sayi1 + sayi2);

}

else if (islem == '-')

{

Console.WriteLine(sayi1 - sayi2);

}

else if (islem == '/')

{

Console.WriteLine(sayi1 / sayi2);

}

else

{

Console.WriteLine(sayi1 \* sayi2);

}

ya da

switch (islem)

{

case '+':

Console.WriteLine(sayi1 + sayi2);

break;

case '-':

Console.WriteLine(sayi1 / sayi2);

break;

case '/':

Console.WriteLine(sayi1 / sayi2);

break;

default:

Console.WriteLine(sayi1 \* sayi2);

break;

ya da

int sonuc = islem switch

{

'+' => sayi1 + sayi2,

'-' => sayi1 - sayi2,

'/' => sayi1 / sayi2,

\_ => sayi1 \* sayi2

};

ya da

Console.WriteLine(islem == '+' ? sayi1 + sayi2 : (islem == '-' ? sayi1- sayi2 : (islem == '/' ? sayi1 / sayi2 : sayi1 \* sayi2)));

//ile tek satırda da halledebiliriz.

Örnek:

// Girilen sayının değeri 10 değilse ekrana 'Sayı yanlış' yazalım.

if (sayi != 10)

{

CW("Sayı yanlış."); // uzun uzun if else yazmaya gerek yok. Daha kısa yazabilmek varken!

}

ya da tek satırda;

Console.WriteLine(sayi == 10 ? "" : "Sayı yanlış."); // sayi 10 ise boş değer, değilse "Sayı yanlış."

Örnek:

// Verilen sayının negatif ya da pozitif olduğunu gösteren uygulama.

int sayi = int.Parse(Console.ReadLine());

// bazı zamanlar akış kontrol bloklarında elde edilen değeri, bloğun dışında da kullanmamız gerekirse bunu bir değişkenle taşımamız gerekebilir.

string sonuc = "";

if (sayi < 0)

sonuc = "Negatif";

else

sonuc = "Pozitif";

Console.WriteLine(sonuc); // -5 girersek ekranda : -5

// Negatif yazacak.

**DERS-6: C# 7.0 Pattern Matching - Type Pattern**

**Type Pattern**

- Object içerisindeki bir tipin belirlenmesinde kullanılan is operatörünün desenleştirilmiş halidir.

- is ile belirlenen türün direkt dönüşümünü sağlar. is'in desenleşmiş halidir.

Örnek:

object x = 125;

if(x is string)

{

string xx = x as string; //string'e dönüştür.

CW($"x değişkeni string tipindedir.");

}

else if(x is int)

{

int xx = (int) string; //int'e dönüştür.

CW($"x değişkeni int tipindedir.");

}

//object içerisinde string varsa if bloğu işlemlerini yap, int varsa else if'i yap.

//ilgili türün hangisi olduğunu runtime'da kontrol edip bulduktan sonra o objectin içerisindeki değeri unboxing yaparken ya //-as- ile ya da cast() ile bir şekilde unboxing yapmamız gerek. Bunu uzun uzun yapmak yerine;

- Desen'i kullanabiliriz.

if (x is string xx) // x eğer ki string ise xx değişkeni ile string olarak geri göndür. (as veya cast kullanmadan)

CW($"x değişkeni string tipindedir.");

else if (x is int xx) // x eğer ki int ise xx değişkeni ile int olarak geri göndür.

CW($"x değişkeni int tipindedir.");

Örnek:

object x = "Ali";

if(x is string)

{

string x = (string)x;

}

ya da

object x = "Ali";

if(x is string a) // eğer ki x'in içindeki değer string ise a değişkeninde string olarak ver.

{

//artık a'yı burada istediğimiz gibi kullanabiliriz.

}

//scope dışında da erişebiliriz ama null olma ihtimalinden dolayı kullanamayız. Bir değişken null ise çağırılamaz, //değeri çağırılamaz sadece ona bir değer atanabilir. x'in string olmama durumunda null gelme şansı var.

//CW(a); //hata verir. ama önce a'ya bir değer atarsak.

a = "Velican";

CW(a); // Hata vermez. Sonuç ekrana yazdırılır.

// Type Pattern kullanıp bir daha else if ile dönüşüm yapacaksak değişken isimleri aynı olmamalı.

if(x is string a)

{

....

}

else if (x is int a) //olmaz hata verir.

{

....}

**C# 7.0 Pattern Matching - Constant Pattern**

- Elimizdeki veriyi sabit bir değer ile karşılaştırabilmemizi sağlar. =='tir operatörü gibi çalışır.

Örnek:

object x = 123;

if(x is 123) //değersel bazlı kontrol

{

}

if(x is int) // tür bazlı bir kontrol

{

}

Örnek:

int a = 123;

CW(a is int); -> ekrana True yazdıracaktır.

- is operatörü bir değişkenin türünü sormamızı/belirlememizi sağlyan bir operatördür. Ve bu operatörün kullanıldığı değişkenlerin türü illa bir referans türlü olmak zorunda değildir. İstersek değer türlü değişkenlerde de is operatörü kullanılır. Primitive türler de dahil.

Örnek:

int a = 5;

//Mini not: Aşağıdaki iki CW'nin çalışması için object olarak yazmak lazım. Yazmaksak hata verir. zaten int olduğu belli.

//Console.WriteLine(a is int); //buradakiler ise is operatörünün normal işlemidir.

//Console.WriteLine(a is string);

//Constant Pattern kontrolü ise;

Console.WriteLine(a is 5);

//Eğer ki is operatörüyle bir değişkenin değerini == operatörünün sorumluluğuyla kontrol ediyorsak

//buna constant pattern denmektedir. Sonuç ekrana True döner.

**C# 7.0 Pattern Matching - Var Pattern**

- Eldeki veriyi var değişkeni ile elde etmemizi sağlar.

Örnek:

object x = "Türkiye";

if (x is var a) //x'in türü ne olursa olsun var a'ya çıkar. Var verilen değerin türüne (runtime'da!) bürünme işlemini gerçekleştiren bir keyword'dür.

{

Console.WriteLine(a);

}

**NOT!:** var keyword'ü normalde derleyici sürecinde türü belirlerken var pattern'daki var runtime'da türü belirlemektedir.

var b = "Alilo"; //direkt string oldu.

ama var pattern'daki;

if(x is var a) //almış olduğu değeri runtime'da alacağından dolayı türü runtime'da belirlenir.

...

Not: Dynamic runtime'da türü belirlenecek değişkenlerde kullanılır. Fakat biz hem elimizdeki değerin türünü kontrol edip ardından o türü kendi verisinde unboxing edeceksek burada var pattern'i kullanak zorundayız. (x is dynamic a) tarzı bir pattern yoktur. Hata verir.

**C# 7.0 Pattern Matching - Recursive Pattern**

- switch case yapılanması üzerinde yenilikler getirmektedir.

- switch bloğunda referans türlü değişkenlerde kontrol edilebilmektedir.

- Ayrıca switch bloğuna when komutu ile çeşitli şart/koşul niteliği kazandırılmıştır.

- Tür kontrolü yaptığı için type pattern'ı kapsamaktadır.

- Ayrıyetten case null komutu ile ilgili referansın null olup olmamasını kontrol edebilmesinden dolayı constant pattern'ı da kapsar.

Detaylar ileride.

**C# 7.0 Pattern Matching - Type ve Var Pattern Üzerine Devam (Genel Kültür)**

- Akış kontrol mekanizması olmadan da kullanabilmekteyiz.

Type Pattern;

object x = "Alilocan";

bool result = x is string o; //hem sonucu bool olarak result'a aldım. Hem de o isimli bir değişken tanımlamış oldum. Ve ilgili string x değerinin orjinal değerini o'ya string olarak cast etmiş olduk.

CW(o); //Hata verir. -o- 'yu null'dan çıkarmak için değer vermek lazım. Bu seferde x'in orjinal değeri bozulur.

//x'in olmadığı durumda o'nun null olma durumu var. Bu ihtimalden dolayı kullanamayız.

// Kullanışlı değil. Type pattern'ı if ile kulanmaya devam.

Var Pattern;

object x = "Alilocan";

bool result = x is var o; //runtime'da null olup olmayacağı kararlaştırılacağı için hata vermez.

CW(o); //hangi tür olursa olsun atayacağından dolayı null gelme ihtimali yok.

**C# 9.0 Pattern Matching Enhancements**

**Simple Type Pattern C# 9.0**

- Normal type pattern'ın geliştirilmiş halidir.

- Bir değişken içerisindeki değerin belirli bir türde olup olmadığını hızlı bir şekilde kontrol etmemizi sağlayan desendir.

- 9.0'dan önce Type Pattern ile yapılan tür bildirimlerinin yanına değişken adı tanımlaması yahut discard ifadesinin kullanılması zorunluydu.

9.0 ile bu ortadan kalktı ve direkt olarak tür kontrol işlemine odaklanılması sağlandı.

Örnek: //sadece tür kıyaslaması yapacaksak

//Eskiden

object obj = new Person();

switch(obj)

{

case Person p:

//.....

break;

}

//9.0 ile

object obj = new Person();

switch(obj)

{

case Person: //p'yi silebiliriz.

//.....

break;

}

Örnek:

//ESKİ

string GetProduct(IProduct p) => p switch

{

Technologic \_=> "Teknolojik",

Computer \_=> "Bilgisayar",

Goggles \_=> "Gözlük"

};

**NOT:**

\_ Discard Operatörü: Bir veriyi, değişkeni, propertyi yahut metot geri dönüş değerini bir noktaya atama işlemi neticesinde ilgili değerin bellekte bir yer ayrılmayacak şekilde çöp olarak kabul edilmesini sağlamakla görevlidir.

Bir başka deyişle değişken deklare(tanımlama) etmeye ihtiyaç duyulmadığı durumlarda Discard operatörü kullanılarak sonraki adımlarda hiçbir zaman kullanılmayacak değişkenleri boşuna oluşturarak bellekte lüzumsuz yer işgal edilmemesini sağlamaktadır.

//Yeni

string GetProduct(IProduct p) => p switch

{

Technologic => "Teknolojik", //herhangi bir değişken ismi ya da discard operatörü vermeden tür kontrolü yapabilmekteyiz.

Computer => "Bilgisayar",

Goggles => "Gözlük"

};

**Relational Pattern C# 9.0**

- Desenlerde <,>, <= ve >= operatörleri kullanabilmekte ve belirli karşılaştırmaları hızlıca gerçekleştirebilmektedir.

Örnek:

int number = 111;

string result = number switch

{

< 50 => "50'den küçük",

> 50 => "50'den büyük,

50 => "50'ye eşit",

\_ => "Hiçbiri

};

//switch özü itibarıyla sadece eşitlik durumunu inceleyebiliyorDU. Relational pattern ile artık diğer türlü durumları da kullanabiliriz.

//if ile farkı kalmamıştır.

**Logical Pattern C# 9.0**

- and, or ve not gibi mantıksal operatörleri keyword olarak kullanabiliriz.

Örnek:

string GetProduct(IProduct o) => p switch

{

Technologic or Computer => "Teknolojik,

//Goggles => "Gözlük"

},

Örnek:

int number = 60;

string result = number switch

{

> 10 and < 50 => "10'dan büyük 50'den küçük",

> 50 or < 100 and 60 => "50'den büyük veya 100'den küçük ve 60'a eşit",

not 51 => "51 değil."

};

- Relational pattern ile oldukça uyumludur.

**Not Pattern C# 9.0**

- not operatörünün kullanılbildiği bir desendir. (bir üstteki örnekte relational pattern'da kullandık.)

Örnek:

object obj = new Goggles();

if (obj is not Technologic)

{

// object bir Technologic değilse (-not-) buraya gir. is ile de kullanılabilir.

}

**DERS-7: Hata Kontrol Mekanizmaları Nedir? Ne Amaçla Kullanılır?**

**Hata Türleri - Derleme/Syntax/Sözdizimi Hatası**

- Program dili kurallarına aykırı olan hatalardır.

- Gelişmiş editörler sayesinde derleeye gerek kalmadan bile farkedebilmekteyiz.

- Hatanın bulundığı satır derleyici tarafından rapor edilir.

- O yüzden fark edilmesi ve çözülmesi en kolay hata türüdür.

Örnek:

stat\_ic void Main(string[] args) //stat\_ic olmaz.

{

...//ya da türlü yazım hataları, uygunsuz scope'lar, tanımlanmayan değişkenler.

}

**Hata Türleri - Run-Time(Çalışma Zamanı) Hataları**

- Syntax'da problem yokken çalışma zamanında hataya sebep verir.

- Yazılım ayaktayken ortaya çıkan bir takım aykırı durumlardan dolayı programın işletim sistemi tarafından kesilmesi/durdurulmasıdır.

- Programın işleyişinin ortasında direkt kullanıcı ile temas edebilecek hatalardır.

Runtime Error!

- Çalışma zamanı hataları genellikle kullanılan mimaride öntanımlı mesajları verecektir.

- Eğer ki alınan hatanın mimaride bir karşılığı yoksa işletim sistemi dili kullanılacak ve daha kompleks bir açıklamayla karşılaşacaktır. Bu yüzden olması gereken çalışma zamanı hatalarının önceden tespit edilip daha anlaşılabilir şekilde düzenlenmesi lazım.

- Bu hataları öngörebilmek genelde -tester-'ların işidir.

- Uygulama mümkün mertebe test edilerek çalışma zamanı hataları tespit edilmesi ve programcı tarafından tanımlanmalıdır.

- Bir programın gelişimi sahadaki belli bir sürece bağlıdır.

- Bu süreçte toplanan log'lar ve son kullanıcı dönütleri değerlendirilerek çalışma zamanı hataları tespit edilip arındırılabilir.

- Tespit edilen çalışma zamanlarının manipülasyonları önemlidir.

- Hata kontrol mekanizmaları "çalışma zamanı hatalarını" kullanıcıya hissettirmeden yakalayabilmek ve ilgili hatayı manipüle etmek için vardır.

Örnek:

//Çalışma zamanı hata durumları

- Olmayan bir dosyayı açmaya yahut üzerine yazmaya, okumaya çalışmak.

- Olmayan değer üzerine işlem yapmak.

- Uygun olmayan formatlarda çalışmak.

- Veritabanı bağlantısının kopması.

Örnek:

using System;

namespace OrnekProje

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Lütfen birinci sayıyı giriniz: ");

int sayi1 = int.Parse(Console.ReadLine());

Console.WriteLine("Lütfen ikinci sayıyı giriniz: ");

int sayi2 = int.Parse(Console.ReadLine());

//integer'a Parse edilemeyecek bir değer girilirse runtime error verir. X -> Exception Unhandled ....

Console.WriteLine("Toplam : " + (sayi1 + sayi2));

//uygulamayı dosya yolundan bulup .exe üzerinden çalıştırırsak daha net görebiliriz.

//sayi1'e "Ali" girersek yine hata verir. "Ali" int'e dönüşmez.

}

}}

- Buna yazılımsal bir çözüm getirmek gereklidir.

**try - catch Mekanizması Teorik Anlatım**

- Çalışma zamanında alınan olası hataları kontrol etmemizi, karşılamamızı, manipüle etmemizi sağlayan bir yapılanmadır.

- try catch yapılanması uygulama sürecinde yaşanan olası hatayıkullanıcıya hissettirmeksizin farklı bir durum ya da olağan bir mesaj gibi göstermenizi sağlayan ve bunun yanında hataya dair bizlere bilgi sunan ve bu bilgiler eşliğinde kayıtlar/log oluşturmamızı sağlayan bir programatik yapılanmadır.

- Amaç:

1)Kullanıcıya alınan hatayı hissettirmemek.

2)Alınan hatanın nedenine dair kullanıcıyı bilgilendirmek.

3)İşletim sistemleri aykırı durum yaşandığında uygılamayı sonlandırmak isterler ve sonlandırırlar. try-catch yapılanması ile alınan hataya dair manipülasyon

gerçekleştiriliyor ve uygulamanın kapanmadan devam edilmesi sağlanabiliyor.

Prototip:

try

{

... //olası çalışma zamanı hatalarını verebilecek kodları buraya yazıyoruz.

}

catch

{

... //try içerisinde olası hata söz konusu ise kodun akışı orada kesilecek ve akış catch bloğundan devam edecektir.

//hataya dair log, kullanıcı bilgilendirme, kontrollü kapanı vs.

}

Örnek:

using System;

namespace OrnekProje

{

class Program

{

static void Main(string[] args)

{

try

{

Console.WriteLine("Lütfen birinci sayıyı giriniz: ");

int sayi1 = int.Parse(Console.ReadLine());

Console.WriteLine("Lütfen ikinci sayıyı giriniz: ");

int sayi2 = int.Parse(Console.ReadLine());

Console.WriteLine("Toplam : " + (sayi1 + sayi2));

}

catch

{

System.Console.WriteLine("Lütfen doğru bir ifade giriniz.");

//değerleri hatalı girdiğimizde burası ekrana yazılacaktır.

}

}

}}

**Not:** try catch yapılanması olası hatanın ihtimal olduğu kodları durmadan kontrol eden maliyetli bir yapıdır. Dolayısıyla try içerisinde kontrol edilen kodlar lüzumsuz yere tüm kodlar olmamalıdır. Sadece olası hata barındıran kodları try-catch'de bulundurmak daha verimli ve performanslı olacaktır.

...

static void Main(string[] args)

{

Console.WriteLine("Lütfen birinci sayıyı giriniz: ");

int sayi1 = 0, sayi2 = 0;

try

{

sayi1 = int.Parse(Console.ReadLine());

Console.WriteLine("Lütfen ikinci sayıyı giriniz: ");

sayi2 = int.Parse(Console.ReadLine());

}

catch

{

Console.WriteLine("Lütfen doğru bir ifade giriniz.");

//değerleri hatalı girdiğimizde burası ekrana yazılacaktır.

}

if(sayi1 is not 0 && sayi2 is not 0)

Console.WriteLine("Toplam : " + (sayi1 + sayi2));

}}} //gibi sadeleştirebiliriz.

**try - catch Hata Parametreleri**

- Çalışma zamanında alınan hataya dair bizlere bilgi veren/taşıyan parametrelerdir.

Örnek:

int s1 = 0, s2 = 10;

int a = s2 / s1;

Runtime hatası verecektir:

Unhandled exception. System.DivideByZeroException: Attempted to divide by zero.

at OrnekProje.Program.Main(String[] args) in C:\Users\acdem\Masaüstü\C#VSCode\OrnekProje\Program.cs:line 25

DivideByZeroException bize kaşılaştığımız hatanın türünü ifade etmektedir.

Bu geri dönüşler hatanın türüne göre değişecektir.

Örnek:

object x = null;

x.ToString();

Unhandled exception. System.NullReferenceException: Object reference not set to an instance of an object.

at OrnekProje.Program.Main(String[] args) in C:\Users\acdem\Masaüstü\C#VSCode\OrnekProje\Program.cs:line 26

Örnek:

int.Parse("Alican");

Unhandled exception. System.FormatException: Input string was not in a correct format.

**try-catch Exception ile Hata Yakalama**

try

{

}

catch(Exception herhangiBirIsim)

// Exception, bizlere fırlatılan hatayla ilgili tüm bilgileri getirecek olan (yani hataları karşılayabilen bir üst türdür.

//herhangiBirIsim parametresi üzerinden bizler alınan hataya dair bilgiler edinebilmekte ve gerekli log'lama vs. operasyonları //gerçekleştirebilmekteyiz.bu parametre catch bloğuna yazılmak/tanımlanmak zorunda değil. Eğer ki tanımlama yapılırsa //hataya dair bilgi alabiliriz. Tanımlama yapılmazsa hata neticesinde catch çalışacak fakat bilgi almayacağız.

{

…

}

Örnek:

try

{

int s1 = 0, s2 = 10;

int a = s2 / s1;

}

catch(Exception ex)

{

Console.WriteLine("Mesaj : " + ex.Message);

}

Ekrana yazacaktır -> Mesaj : Attempted to divide by zero.

Not: Exception tüm hata türlerinin atasıdır. Bütün veri türlerinin atası da object'tir. Dolayısıyla Exception'un da atası object'tir. Bu da böyle bir not. Nihayetinde Exception referans türlüdür.

**try - catch Exception Dışında Farklı Bir Tür İle Hata Yakalama**

Örnek:

try

{

int s1 = 0, s2 = 10;

int a = s2 / s1;

}

catch(DivideByZeroException ex) //catch bloğu bir parametre tanımlanmazsa tüm hataları karşılayabilen bir bloktur. //Exception durumunu da anlattık, eğer ki parametre Exception değil özelleştirilmiş bir hataya indirgenmişse böyle bir //durumda sadece ilgili türe ait hataları yakalayabilecek, karşılayabilecektir.

{

Console.WriteLine("Mesaj : " + ex.Message);

}

Örnek:

try

{

int s1 = 0, s2 = 10;

int a = s2 / s1;

int.Parse("Alican");

}

catch(DivideByZeroException ex) //Bu senaryoda try catch kullandğımız halde kod patlayıp sona ermiştir.

//catch bloğu genel olmayan bir türe özgüyse böyle bir durumda sadece o türden hataları yakalayabilir.

{

Console.WriteLine("Mesaj : " + ex.Message);

}

Birden fazla catch bloğu ile bunu çözebiliriz.

try

{

int.Parse("Alican");

int s1 = 0, s2 = 10;

int a = s2 / s1;

}

catch(DivideByZeroException ex)

{

Console.WriteLine("Mesaj : " + ex.Message);

}

catch(FormatException ex)

{

Console.WriteLine("Mesaj : " + ex.Message);

}

catch(Exception ex)

{

//catch bloklarının en sonuna Exception türünde parametre(catch) tanımlarsan alınan hata üstteki türlerden herhangi biri değilse bu Exception tarafından karşılanabilir.

Console.WriteLine("Mesaj : " + ex.Message);

}

- try-catch yapılanması olas runtime hatası alındığında catch sıralamasına göre kontrol etmektedir. Bu sıralama önemlidir. catch Exception'u en sona yazmalıyız.

**try - catch finally Bloğu**

- finally bloğu, try catch yapılanmasında hata alınsa da alınmasa da her iki durumda da tetiklenmesi/çalıştırılması gereken kodları yazdığımız bloktur.

try

{

//hata verebilecek kodlar

}

catch

{

//hata alındıktan sonra yapılacak işlemler

}

finally

{

//her iki durumda da çalışmasını istediğimiz kodlar.

}

- Günümüzde pek bir kullanımı yok. Eskiden veri tabanı bağlantısını kesmekte kullanılıyordu.

**try - catch when Yapısı İle Hata Filtreleme (C# 6.0)**

try

{

int s1 = 0, s2 = 10;

int a = s2 / s1;

}

catch(\_\_\_\_\_) when (\_\_\_karşılaştırma\_\_\_) // catch when(\_\_\_\_) olarak da yazılabilir. when(3==3), when(x == "b") ise gibi.

{

Console.WriteLine("Mesaj : " + ex.Message); //catch uyduğu VE karşılaştırma true olduğu sürece çalışır.

}

Örnek:

string x = "b";

try

{

int s1 = 0, s2 = 10;

int a = s2 / s1;

}

catch(DivideByZeroException ex) when (x == "a") //string x = "a"; olsaydı buraya

{

Console.WriteLine("Mesaj : " + ex.Message);

}

catch(DivideByZeroException ex) when (x == "b") //string x = "b"; olduğu için buraya

{

Console.WriteLine("Mesaj : " + ex.Message);

}

catch(Exception ex) // //string x = "c"; olsaydı buraya. Ne olur ne olmaz diye eklemek lazım.

{

Console.WriteLine("Mesaj : " + ex.Message);

}

**Hata Türleri - Mantıksal Hatalar - BUG**

- Program mantığında, akışında, algoritmasında, stratejisinde bir takım şeylerin yanlış hesaplanması, düşüünülmesi, tasarlanması neticesinde alınan hatalardır.

- Kod çalışır, sonuç verir ama istenilen sonuçlar verilmez.

- Test sürecinde veya müşteri geri dönüşlerinde tespit edilebilir.

- Günlük hayattki karşılığı 'Bug' dur.

- Bazen tek çözüm debug'dır.

Örnek: //2 ile 5'i çarp

CW(2\*6);

Sonuç: 12

Bu kadar basit. Ama sinir bozucu!

Örnek:

bool medeniHal = true;

if (!medeniHal)

{

CW("Hediye gönder.");

}

else

{

CW("Hediye gönderme.");

}

//Yine ufak basit bir hata. Tam tersi olmalıydı.

**Örnek:**

//Kullanıcıdan aldığı dönüşe göre işlem yapan Mini ATM

//1- Bakiye Görüntüleme

//2- Para çekme

//3- Para Yatırma

//4- q'ya basarsa çıkış yapılsın.

using System;

namespace OrnekProje

{

class Program

{

static void Main(string[] args)

{

System.Console.WriteLine("ATM'ye Hoşgeldiniz.");

System.Console.WriteLine("Yapmak istediğiniz işlemi seçiniz.");

int bakiye = 1000;

string secim = Console.ReadLine();

if (secim == "1")

System.Console.WriteLine("Şu anki bakiyeniz: " + bakiye);

else if (secim == "2")

{

System.Console.WriteLine("Çekmek istediğiniz tutarı giriniz: ");

int cekilecekTutar = Convert.ToInt32(Console.ReadLine());

if(cekilecekTutar <= bakiye)

{

System.Console.WriteLine(cekilecekTutar + " TL başarıyla çekildi.");

System.Console.WriteLine("Kalan tutar: " + (bakiye-cekilecekTutar));

}

else

System.Console.WriteLine("Yetersiz bakiye.");

}

else if (secim == "3")

{

System.Console.WriteLine("Yatırmak istediğiniz tutar: ");

int yatirilacakTUtar = Convert.ToInt32(Console.ReadLine());

System.Console.WriteLine(yatirilacakTUtar + " TL başarıyla yatırıldı.");

System.Console.WriteLine("Yeni bakiyeniz: " + (bakiye+yatirilacakTUtar));

}

else if (secim == "q")

{

System.Console.WriteLine("ATM'den çıkış yapıldı. İyi günler.");

}

else

System.Console.WriteLine("Lütfen geçerli bir değer giriniz!");

}

}

}

**Örnek:** //Beden Kitle İndeksi Bulma Uygulaması

//Beden Kitle İndeksi = Kilo / (boy\*boy)

//18'e eşit veya küçükse ZAYIF

//18-25 arası NORMAL

//25'ten büyükse OBEZ yazdıran bir program yazdıralım.

System.Console.WriteLine("Kilonuzu giriniz: ");

int kilo = Convert.ToInt32(Console.ReadLine());

System.Console.WriteLine("Boyunuzu giriniz: ");

double boy = Convert.ToDouble(Console.ReadLine());

double bke = kilo / (boy\*boy);

if (bke<18)

System.Console.WriteLine("Değeriniz." + bke + "=> Normal indeks değerinin altındasınız.");

else if (bke > 18 && bke < 25)

System.Console.WriteLine("Değeriniz." + bke + "=> Normal değerlerdesiniz.");

else if (bke > 25)

System.Console.WriteLine("Değeriniz." + bke + "=> Normal değerlerin üstündesiniz.");

//Boy değerini girerken virgülle 1,68 gibi girmemiz gerekli.

**Örnek:** Switch Case ile Mini ATM Uygulaması

int bakiye = 700;

System.Console.WriteLine("ATM'ye hoşgeldiniz.");

System.Console.WriteLine("Lütfen bir işlem seçiniz.\n\n 1-Bakiye Görüntüle.\n 2-Para çek.\n 3-Para Yatır.\n 4-Çıkış yap.\n");

string secim = Console.ReadLine();

switch(secim)

{

case "1":

System.Console.WriteLine("Bakiyeniz: " + bakiye);

break;

case "3":

System.Console.WriteLine("Yatırmak istediğiniz miktarı giriniz: ");

int yatirilacakTUtar = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("Yeni bakiyeniz: " + (bakiye + yatirilacakTUtar));

break;

case "2":

System.Console.WriteLine("Çekmek istediğiniz miktarı giriniz: ");

int cekilecekTutar = Convert.ToInt32(Console.ReadLine());

if(cekilecekTutar > bakiye)

System.Console.WriteLine("Yetersiz bakiye.");

else

System.Console.WriteLine("Kalan bakiyeniz: " + (bakiye - cekilecekTutar));

break;

case "4":

System.Console.WriteLine("Hesabınızdan çıkış yapılıyor.");

break;

default:

System.Console.WriteLine("Lütfen geçerli bir işlem numarası giriniz.");

break;

}

**Örnek:**

Ekrana aşağıdakini yazdıralım.

1

12

123

1234

12345

123456

1234567

12345678

123456789

for (int i = 1; i <= 10 ; i++) //i=1 ile alttaki döngüye girdik.

{

for(int j = 1; j <= i ; j++)

//j değerim i'ye kadar yani 1'e kadar dönecek ve ekrana j'yi yani 1'i yazacak. j bir artıp 2 olacak ve 2 <= 1 bozulup içteki döngüden çıkacak

{

System.Console.Write(j);

}

System.Console.WriteLine("");

//iç döngüden çıkıp buraya geldik. "" ifadesi alt satıra geç anlamına da gelir. Alt satıra geçtikten sonra üstteki döngüden devam edecek. i++ ile i'yi 2 yapacak.

}

**Örnek:**

while döngüsü ile 100'e kadar olan çift sayıları ekrana büyükten küçüğe yazdıralım.

int a = 100;

while(a >= 0)

{

System.Console.WriteLine(a);

a-=2;

}

**Örnek:**

do while ile 1'den 100'e kadar olan sayıların toplamını ekrana yazdıralım.

int a = 1, toplam = 0;

do

{

toplam += a;

a++;

}

while(a <= 100);

System.Console.WriteLine(toplam);

**Örnek:**

Çarpım tablosu:

static void Main(string[] args)

{

for(int i = 1; i <= 10 ; i++)

for(int j = 1; j <= 10; j++)

System.Console.WriteLine(i + "x" + j + "=" + (i\*j));

}

**Kullanıcı giriş sayfası uygulaması.**

Kullanıcı adı ve şifresini doğru girerse "Tebrikler başarıyla giriş yaptınız."

Yanlış girerse hata verelim ve 3 hak tanıyalım, yanlış girdiği sürece döngü devam etsin.

static void Main(string[] args)

{

int hakSayisi = 3;

while(true)

{

System.Console.WriteLine("Lütfen adınızı giriniz: ");

string kullaniciAdi = Console.ReadLine();

System.Console.WriteLine("Lütfen şifrenizi giriniz: ");

string sifre = Console.ReadLine();

if (kullaniciAdi == "Enes" && sifre == "123")

{

System.Console.WriteLine("Tebrikler başarılı bir şekilde giriş yaptınız.");

break;

}

else

{

System.Console.WriteLine("Kullanıcı adı veya şifre hatalı.");

if(hakSayisi > 0)

{

hakSayisi -=1;

}

if(hakSayisi == 0)

{

System.Console.WriteLine("Giriş hakkınız doldu.");

break;

}

}

}

}

**DERS-8: DÖNGÜLER**

**for - while - do while**

"Hangi döngü nere kullanılmalıdır?" sorusu yanlıştır. Doğrusu "Hangi döngü hangi senaryoya daha çok yakışır?" olmalıdır.

- Ardışık denetlemelerde for döngüsü, sonsuz denetlemerde ise while daha uygundur.

- Bu döngülerin hepsi bir kombinasyona bağlı çalıştıkları için en nihayetinde birbirlerinin yerine kullanılabilirler.

**For Döngüsü**

Prosedürel programlamada döngü yapılarından birisidir.

Genellikle ardışık işlemlerde kullanılan bir döngü yapılanmasıdır.

Prototip:

for(\_\_genellikleBaşlangıçDeğeriIsimliDegisken\_\_;\_\_\_Şart\_\_\_;\_\_GenellikleBaslangıçDeğeriniArtırmaAzaltma\_\_)

{

..........//şart true oldukça bu scope tetiklenecek ve döngü çalıştırılmış olacaktır.

}

- Genellikle ifadelerine dikkat edelim.

- Başlangıç değer ya da ilk değer atama kısmı: Ardışık algoritmalarda genellikle bir ilk değere ihtiyaç vardır. Bunu burada tanımlayabiliriz. Fakat bu tanımlama zorunlu değildir. Ayrıca illa ki integer olması da zorunlu değildir. Ama illa ki bir değer atanmalıdır.

- Şart kısmında ise: Herhangi bir şart/koşul ifadesi tanımalanabilir. Genellikle ilk değer olarak tanımlanan değişken durumu burada kontrol edilir. Yani bir şarta bağlanır. Şart true olduğu sürece döngü tetiklenecektir.

- Son kısımda ise herhangi bir değişken üzerinde işlem yapabiliriz. Genellikle başlangıç değeri üzerinde artırma/azaltma işlemi yapılır. Fakat bu zorunlu değildir.

Örnek:

for ( int sayi = 1 ; sayi < 10 ; sayi++ )

{

}

Kodun çalışması:

1- Akış ilk olarak başlangıç değişkeni tanımlama kısmına girecektir. Eğer ki bir değişken tanımlama ifadesi varsa ilgili değişkeni tanımlayacaktır.

2- Ardından koşula gidecek ve koşulu değerlendirecektir.

3- Koşul true ise döngüye girecek,

3.1-Koşul kontrol edildikten sonra döngü ilgili değeri arttıracak parametreye gidecek ve yapılan artimatik işlemi değişkene uygulayacaktır.

3.2-Sayısal değeri arttırılmış ya da azaltılmış olan değişkenden sonra yeniden koşulu kontrol edecek ve ardından true ise 3, değilse 4. adımlar tekrar edecektir.

Koşul false olana kadar buraya kadarki işlemler tekrar edecek ve döngü tetiklenmiş olacaktır.

4- Koşul true değilse döngüden çıkacaktır.

Örnek:

for(int i = 1 ; true ; i++ )

//ya da for(int i = 1 ; i++ )

//ya da for(int i = 1 ; i < 10)

//sonsuza kadar çalışırlar.

Örnek:

for(int i = 1 ; i < 10 ; i++ ) //sonsuza kadar çalışır.

{

CW("Alican");

}

9 kere ekrana alt alta Alican yazdırır. 10 kere yazsın istiyorsak i <=10 demek lazım.

Örnek:

for (int i = 0 ; i < 50 ; i+=5)

{

CW("Hilmi"); //10 kere Hilmi yazacaktır.

}

Örnek:

for (int i = 10; i > 0 ; i--)

{

CW("Hilmi"); //10 kere Hilmi yazacaktır.

}

Örnek:

for (int i = 50; i > 0 ; i-=5)

{

CW("Hilmi"); //10 kere Hilmi yazacaktır.

}

//Eğer ki ardışık işlemlerde artış yapılıyorsa i'nin değeri genellikle < (küçüklük) durumuyla değerlendirilir.

// Eksilme işlemi yapılıyorsa da > (büyüklük) durumu ile değerlendirilir.

Örnek:

//1'den 10'a kadar olan sayılaı alt alta ekrana yazdıralım.

for (int i = 1 ; i <= 10 ; i++)

{

CW(i);

}

ya da azıcık karıştıralım.

for (int i = 50 ; i < 60; i++)

{

CW(i-49);

}

Örnek:

//1 ile 40 arasındaki çift sayıları toplayarak sonucu ekranda gösterelim. 40 dahil.

int toplamSonuc = 0; //her bir döngü tekrarında sürekli tanımlanmaması için dışarıda oluşturduk.

for (int i = 1; i <= 40 ; i++)

{

//ilgili sayıyı 2'ye bölüp kalanı 0 kontrolü ile bulabiliriz.

if( i % 2 == 0)

{

toplamSonuc += i;

}

}

System.Console.WriteLine(toplamSonuc); // 420 yazdırır.

Örnek:

//Klavyeden girilen sayının faktöriyelini bulan programı yazınız.

int sayi = int.Parse(Console.ReadLine()); //try catch ile de kontrol edebiliriz.

int faktoriyel = 1;

for (int i = 1 ; i <= sayi; i++) //1'den sayiya gidelim.

{

faktoriyel \*= i;

}

System.Console.WriteLine("Faktöriyel: " + faktoriyel);

ya da sayi'dan geriye gidelim.

for ( int i = sayi; i > 0 ; i--)

{

faktoriyel \*= i;

}

System.Console.WriteLine("Faktöriyel: " + faktoriyel);

ya da daha da geliştirelim.

int sayi = int.Parse(Console.ReadLine());

string sonuc = "";

int faktoriyel = 1;

for (int i = sayi ; i > 0 ; i--)

{

faktoriyel \*= i;

sonuc += i + (i==1 ? " = " : " x ");

}

System.Console.WriteLine("Faktöriyel: " + sonuc + faktoriyel);

//5 girersek;

//Faktöriyel: 5 x 4 x 3 x 2 x 1 = 120 ekrana yazılır.

**For Döngüsü Varyasyonları**

- Başlangıç değişkenini dışarıda da tanımlayabiliriz.

for (; i < 10 ; i++)

{

}

- Bu varyasyonda dışarıda tanımlanan i değişkenin for döngüsünde başlangıç değeri olarak göstermek istiyorsak eğer, yine bir başlangıç değerini içeride vermemiz gereklidir.

int i = 0;

for (int i = 0; i < 10 ; i++)

{

}

- Arttırma/azaltma işlemini for scope'u içine taşıyabiliriz.

for (; i < 10 ;)

{

i++;

}

- Şart kısmı illa ki başlangıç değerine bağlı olmak zorunda değildir.

string adi = "Ali";

for (int i = 0; adi == "Ahmet" ; i++) //değişkenin adı Ahmet olduğu sürece döngüye girecektir.

{

}

Örnek:

int a = 10;

for( int i = 0; a != i\*2; i++) //i'nın iki katını al sonuç a'ya eşit olmadığı sürece devam et.

{

CW("Ali");

}

Ekrana 5 kere Ali yazacaktır.

- Fantastik işlere de girişilebilir. Aşağıdaki kod sonsuza kadar çalışır/tetiklenir.

for ( ; ; )

{

CW("Ali");

}

Sonsuz sayıda Ali yazar.

- Yine sonsuza kadar çalışan bir kod.

for ( int i = 0 ; ; i++ )

{

}

- Önemli! Birden fazla başlangıç değişkeni tanımlayabiliriz. Birden fazla şart koşabiliriz.

for(int i = 0, i2 = 0; i < 10 && i2 < 5 ; i++, i2++) //i2++, i++ sıralama önemli değil.

{

...

}

**While Döngüsü**

while (\_\_\_\_şart\_\_\_\_) //şart doğrulandıkça tetiklenir.

{

...

}

- while döngüsü sadece şarta bağlı bir döngüdür. for'a kıyasla daha ilkel ve sade bir döngüdür.

- while döngüsü programlamanın ilk tasarlanmış döngüsüdür.

- Genellikle sonsuz döngülerde ve sonucu bilinmeyen durumlarda kullanılır. Yine de istediğimiz yerde kullanmakta özgürüz.

While Döngüsünü For İle Kıyaslayalım

for (int i = 0; i < 10; i++)

{

CW("Alican Velican");

}

-> İki kod aynı anlama gelir.

int i = 0;

while (i<10) //while döngüsünde for gibi çalışabilmek için kombinasyonu kendimiz oluşturmalıyız.

{

//i++; bu örnekte farketmez nereye yazıldığı.

CW("Alican Velican");

i++;

}

- While döngüsü şart false olana kadar tetiklenecektir.

Örnek:

//Ekrana 10 kere "Merhaba Dünya" yazdıran bir program yazalım.

int i = 1;

while (i <= 10)

{

CW("Merhaba Dünya");

i++;

}

Örnek:

//Klavyeden girilen sayıdan geriye doğru 0'a kadar sayan bir sayaç hazırlayalım.

int sayi = int.Parse(Console.ReadLine());

while (sayi >= 0)

{

//sayi--; bu örnekte sorun çıkar. Çünkü ekrana yazacağımız değeri kullanıyoruz.

Console.WriteLine(sayi);

sayi--; //buraya yazmamız lazım.

}

Not: While döngüsünde çalışırken artırma/azaltma operatörlerinin nerede kullanıldığına dikkat etmek lazım.

Örnek:

//while döngüsü ile 0-100 arasındaki tek sayıları toplayarak sonucu ekranda gösteren programı yapalım.

int i = 0, toplam = 0;

while (i <= 100)

{

if (i%2 == 1)

toplam +=i; // toplam += i++; yazamayız. if şartını sonsuza kadar sağlayacağı için olmaz.

i++;

}

Örnek:

//Klavyeden girilen sayının faktöriyelini hesaplayalım.

int sayi = int.Parse(Console.ReadLine());

int faktoriyel = 1;

while( sayi>0 )

{

faktoriyel \*= sayi;

sayi--;

// faktoriyel \*= sayi--; 'yi burada kullanabiliriz. if olmadığından dolayı her durumda sayi değişkeninin değerini önce faktoriyel'e verecek ardından 1 düşürecektir.

}

CW(faktoriyel);

Örnek:

//O anki tarihin saniye değeri 5'in katıysa eğer tarihi ekranda gösteren uygulamayı yazalım.

while(true)

{

if(DateTime.Now.Second % 5 == 0 )

{

Console.WriteLine(DateTime.Now);

}

}

**Do While Döngüsü**

do{

.

.

.

}while(\_\_\_\_şart\_\_\_\_);

- while döngüsü önce şarta bakar true ise kodu çalıştırır. Falsa olursa kod bloğu çalışmaz.

- do while ise önce kodu çalıştırır sonra şarta bakar. Şart true da olsa false da olsa bir kere tetiklenir.

Örnek:

do{

Console.WriteLine("do while döngüsü");

}while(false);

Not: Tek satırda döngülere sokmak istediğimiz kodları scope'suz yazabiliriz.

for ( int i = 0; i < 10; i++)

CW("Alican Mehmetovski");

ya da

while (true)

CW("");

ya da

do

CW("Apaçi Serkan");

while(true);

**Sonsuz Döngüler**

For ile Sonsuz Döngüler

for (int i = 0; true; i++) //int'ın değer aralığının sonuna ulaştığında döngü biter.

{

}

onun yerine;

for (; ;)

{

...

}

yazarsak sonsuz döngüye gireriz.

Örnek:

//bir aksiyonu aradığımızı ve bulduğumuz anda çıkmamız gerektiğini düşünelim.

bool dongu = true; //for'un dışına bağlayıcı bir değişken atamamız lazım.

for (; dongu ;)

{

if(true)

{

dongu = !dongu; //tam tersini donguye atarsak false olup bir sonraki turde sonsuz döngüden çıkarır.

} }

while ile Sonsuz Döngüler

while (true)

{

....

}

Örnek: //mesela mouse tıklanana kadar döngüye devam eden bir kod olsun.

bool durum = false;

while(!durum) // true verseydik direkt durum yazabilirdik.

{

if(true) //burada tıklanmış olsun.

{

durum = !durum; //fantezi aradık. Maksat farklılık.

}

}

Do While ile Sonsuz Döngüler

do{

.

.

} while (true);

Sonsuzluktan çıkmak için ise;

bool durum = true;

do{

if(true)

{

durum = !durum;

}

} while (durum);

İç İçe Döngüler Nelerdir?

for(int i = 0; i < 10; i++)

{

for( int j = 0; j < 5; j++)

{

while(true)

{

...

}

}

}

Örnek:

for(int i = 0; i < 10; i++) //burada i kullandık. 10 kere döner.

{

for(int j = 0; j < 5; j++) //burada farklı bir harf kullanmaya özen göstermeli. 10\*5 kere 50 kere döner.

{

for(int o = 0; o < 3; o++) //yine harf değişelim. 10\*5\*3 150 kere döner.

{

int p = 0;

while(p < 4)

{

p++; // 10 \* 5 \* 3 \* 4 = 600 kere döner.

}

}

}

}

- İç içe döngülerde maliyet tüm döngülerin maliyetinin/tur sayısının/periyodik çalışmasının çarpımına eşittir.

- Dıştaki döngünün 1 turunda alttaki döngü tam turunu yapar.

**Foreach Bir Döngü mü? İterasyon mu?**

- İterasyondur. (Kandırıldık. Döngü değilmiş.)

- Döngü, belirli bir kombinasyon eşliğinde çalışan ve belirli bir şarta bağlı olan periyodik işlemler gerçekleştiren yapılanmadır.

- İterasyon ise mantığında ne kombinasyon ne şart vardır. İterasyondaki sonraki veri, öteki veri anlamına gelen itere etmek fiili vardır. (ileride)

Bir veri kümesi üzerinde işlem yapmamızı/verileri elde etmemizi sağlayan yapılanmadır.

**DERS-9: Keywords**

- Programlama dilinin en atomik parçalarıdır.

- Derleyici için ön tanımlı olan, nerede hangi amaca hizmet edeceği belli ve kod içerisinde hangi noktalarda çağırılabileceği/kullanılabileceği kurallarla sınırlandırılmış olan anahtar sözcüklerdir.

- Keyword'lerin operatörlerden farkı: Operatörler esas olarak belirli bir operasyonu/eylemi üstlenen yapılardır.

Keywordler daha geniş kavramlardır.

- Operatörler esasında bir keyword'dür lakin her keyword bir operatör değildir.

Örnek: var keyword'ü hem pattern matching'de farklı hem de değişken seviyesinde farklı bir kullanıma sahiptir.

Örnek:

namespace OrnekProje //namespace keyword

{

class Program //class keyword

{

static void Main(string[] args)

{

for(...) //for keyword'ü

System.Console.WriteLine("Hello World!");

do //keyword

{

}while(true); //while keyword

}

}

}

- Konseptli keyword'lerin konseptine hakim olmak gerekir. Anlam kazanmak için bir bütüne ait olmasına veya yanına yazılacak konseptlere ihtiyacı vardır. Değişken türleri de bir konseptli keyword'lerdir.

- Konseptsiz keyword'ler de vardır. Sayıları daha azdır.

return;

break;

continue;

goto;

gibi örnekler verilebilir.

**Manevratik Komutlar**

- Normal gidişata müdahele etmek (döngülerden çıkmak, kodu durdurmak veya devamını okumamak, methodu sonlandırmak gibi) için yardımcı olan komutlardır.

Örnek: //23. adımda döngüden çıkmak istersek.

for(int i = 0 ; i < 100; i++)

{

if (i == 22)

{

i = 100;

}

}

//bu şekilde algoritmik bir çözüm getirebiliriz.

//ya da daha kolayı var.

- Manevratik komutlar yapabileceğimiz kod yönlendirmelerini daha efektif yapmamızı sağlar.

**break komutu**

- switch case yapılanmasında ve döngülerde kullanılır. Sadece bu iki yapı içerisinde kullanılan bir keyword'dür.

- Kullanıldığı yapıyı sonlandırmaya yarar.

- İç içe döngülerde en içteki döngüde kullanılıyorsa sadece onu sona erdirir.

Örnek:

for(int i = 0 ; i < 100; i++)

{

while (true)

{

break;

}

}

Örnek:

do

{

if(true)

{

break; //if bloğu döngünün içerisinde olduğu için burada da break'i kullanabiliriz.

}

}while(true);

Örnek:

Saniye 45 olduğunda DateTime'dan çıksın.

while(true)

{

if(DateTime.Now.Second == 45)

break;

Console.WriteLine(DateTime.Now);

}

Çıktı:

.

20.09.2022 17:06:44

20.09.2022 17:06:44

20.09.2022 17:06:44

20.09.2022 17:06:44

20.09.2022 17:06:44 //45'e gelince çıkar.

Örnek:

for(int i = 0; i < 5; i++)

{

for(int j = 0; j < 3; j++)

{

if(j == 1)

break;

Console.WriteLine("i : " + i + " j : " + j);

}

}

Çıktı:

i : 0 j : 0 //dıştaki döngü her sarmaya başladığında içteki j'nin 1 olduğu durumda içteki döngüden çıkış yapıp dıştaki döngüye i'yi arttırarak devam edecek.

i : 1 j : 0 //i artacak ama j hep sıfır kalacak.

i : 2 j : 0

i : 3 j : 0

i : 4 j : 0

Örnek:

Kullanıcıdan 't' harfi girilene kadar alınan sınırsız sayıda sayıyı toplayan ve sonucu ekrana yazdıran uygulama.

using System;

namespace OrnekProje

{

class Program

{

static void Main(string[] args)

{

int toplam = 0;

while(true)

{

System.Console.WriteLine("Lütfen bir sayı giriniz.");

string girilenDeger = Console.ReadLine();

if(girilenDeger == "t")

{

System.Console.WriteLine("Toplam Sonuç: " + toplam);

break; //ekrana tekrar üstteki satırın yazılmasını istemiyorsak kodu burada sonlandırmamız lazım.

}

else

{

toplam += int.Parse(girilenDeger);

}

}}}}

Örnek:

Kullanıcıdan alınan sonsuz adet sayı değerlerinden 37'nin katı girildiğinde sonlanan uygulamayı yazalım.

using System;

namespace OrnekProje

{

class Program

{

static void Main(string[] args)

{

while (true)

{

System.Console.WriteLine("Lütfen bir sayı giriniz.");

int sayi = int.Parse(Console.ReadLine());

if (sayi % 37 == 0)

{

System.Console.WriteLine("Uygulama sonlanmıştır.");

break;

}

}}}}

**continue komutu**

- Sadece döngülerden erişilebilir ve kullanılabilir. "Döngü içindeki" döngü olmayan yapılanmalarda kullanılabilir.

- Amaç döngüde bir sonraki tura direkt geçilmesini sağlar.

Örnek:

for (int i = 0; i < 10; i++)

{

if(i%2 != 0) //i'nin 2'ye bölümünden kalan sıfır değilse devam et. Bir sonraki tura geç.

continue;

Console.WriteLine(i);

//yani çift olmayan durumlarda buradaki işlemi yapmaz. Sadece çift olduğu durumlarda bu satır çalışır.

}

Örnek:

//Kullanıcının girdiği sonsuz adet sayıdan pozitif olanlarını çarpan ve 't'(enter) yapıldığında sonucu ekrana yazdıran kodu yazalım.

using System;

namespace OrnekProje

{

class Program

{

static void Main(string[] args)

{

int carpim = 1;

while(true)

{

System.Console.WriteLine("Lütfen bir sayı giriniz.");

string girilenDeger = Console.ReadLine();

if (girilenDeger == "t")

{

System.Console.WriteLine(carpim);

break;

}

else //else bloğunu koymasak da olur. Kod aynı şekilde çalışır. Ek bilgi.

{

int sayi = int.Parse(girilenDeger);

if(sayi < 0)

continue; //sayı negatifse bekleme yapma geç. Sonraki tura geç.

carpim \*= sayi;

}

}

}}}

Sonuç:

Lütfen bir sayı giriniz.

45

Lütfen bir sayı giriniz.

78

Lütfen bir sayı giriniz.

10

Lütfen bir sayı giriniz.

t

35100

Örnek:

//1 ile 1000 arasında 7'nin katı olmayan sayıları ekrana yazdıralım.

using System;

namespace OrnekProje

{

class Program

{

static void Main(string[] args)

{

for (int i = 1; i < 1000; i++)

{

if (i % 7 == 0)

continue; //7'nin katı olanları geç. Döngüye sonraki adımdan devam et.

System.Console.WriteLine(i);

}}}}

**return Komutu**

- Metod içerisinde her yerde kullanılabilir, erişilebilir bir keyword'dür.

Main'de bir method yapılanmasıdır. Yani main içerisinde her yerde kullanılabilir.

- return iki işleme yarar.

1) Nerede çağırılıyorsa çağırılsın bulunduğu metoddan çıkış yapar. Yani sonlandırır. return'den sonra hangi komut geliyorsa gelsin işlenmez.

2) İleride göreceğimiz metodlar konusunda geriye değer döndürme sorumluluğunu üstlenen bir keyworddür.

Örnek:

//Kullanıcı 'c' tuşuna basana kadar sonsuz döngüde dönen uygulamayı yazınız.

//readLine enter ile çalışır. O yüzden readKey kullanmak lazım.

using System;

namespace OrnekProje

{

class Program

{

static void Main(string[] args)

{

while(true)

{

//ReadLine enter'a basıldığında çalışır.

if(Console.ReadKey().KeyChar == 'c')

{

System.Console.WriteLine("Uygulama sona ermmiştir.");

return;

}

System.Console.WriteLine("");

System.Console.WriteLine("Uygulama çalışıyor.");

}}}}

Sonuç:

4

Uygulama çalışıyor.

q

Uygulama çalışıyor.

w

Uygulama çalışıyor.

<

Uygulama çalışıyor.

Uygulama çalışıyor.

cUygulama sona ermmiştir.

**goto Komutu (en sevilmeyen keyword)**

- Kodun senkronizasyonunu bozup, akışı ters istikamette almamızı sağlayan bir manvratik komuttur.

- Davranışsal olarak döngülere benzer.

- switch case yapılanmasında dahili olarak kullanılan bu komut metod içerisinde her yerde kullanılabilir.

switch(\_\_\_\_)

{

case n:

break;

case m;

goto case n;

}

Prototip:

a: //başlangıç referansı

.

.

.

goto a; // tanımlanmış olan referansa kodun akışını yönlendiriyoruz.

- Geleneksel coder'lar tarafından pek sevilmez. Tavsiye edilmez.

- Teknik olarak programı yavaşlattığı söylenir. Performans testlerinde kayıp ve yavaşlık olduğu görülmüştür. Maliyeti diğer durumlara göre fazladır.

- goto keyword'ü ile senkronizasyonu bozup basa dönme durumu bir döngüyle aynı işlemi yapmaya nazaran daha maliyetli olacaktır.

- Gerekmediği sürece kullanmamak lazım.

Örnek:

a:

CW("Merhaba");

CW("Dünya");

goto a;

//Sonsuz döngü şeklinde çalışmaya devam eder. Sürekli ekrana yazdırır.

//Böyle bir durumda önerilen "döngü" kullanmaktır.

Örnek:

int i = 1;

x:

Console.WriteLine(i++);

if(i <= 100) //true olduğu sürece x'ye gidecek.

goto x;

- goto keyword'ü belirtilen yere geri dönüş yaparken scope kavramını dinlemez.

a:

for (int i = 0; i < 100; i++)

{

goto a; //scope dışındaki a'ya geri dönüş yapar. i'yi arttırmaz sonsuza kadar i'ye sıfır değerini atar.

}

**Ekstra Bilgiler**

- Döngülerde boş scope kullanmak istemiğimiz durumlarda ; noktalı virgül operatörü ile temiz kod yazımı yapabiliriz.

while(true)

{

}

yerine

while(true);

for ( ; ; )

{

}

yerine

for ( ; ; );

do

{

} while (true);

yerine

do;

while(true);

Ve bu şekilde sonsuz döngüleri daha kısa formatta yazabiliriz.

- Bazen (ileride asenkton programlamada) içi boş sonsuz döngülerde ihtiyacımız olabilir.

- Bu durumlarda normal döngü gövdelerini kullanabileceğimiz gibi scope'suz bu şekilde de kullanabiliriz.

Aynı şekilde if scope'unda da benzer kullanım uygulanabilir.

if (true)

{

}

Yerine if (true); //yazabiliriz. Çok karşımıza çıkmaz.

**DERS-10: ARRAYS - DİZİLER**

- Tek bir değişken altında birden fazla "aynı türde" değeri toplamamızı sağlayan veri kümelerine dizi denmektedir.

- Prosedürel programlamanın temel veri kümeleridir. Yani yazılımsal boyutta, yazılım adına RAM'de bbirden fazla geğeri

tek bir değişken altında bir veri kümesi halinde tutabilirler.

- Diziler, veri kümeleri oldukları için, içlerindeki birden fazla veri üzerinde kümesel işlemler yapmamızı sağlayabilirler.

- Diziler, prosedürel programlamanın temel yapıları oldukları için daha gelişmiş yapılar olan koleksiyonlarında temelini

teşkil etmektedirler ve gelişmelerine katkıda bulunmaktadırlar.

- Diziler, referans türlü değerlerdir. Yani nesnel yapılardır. Özlerinde class'tırlar.

- Yapısal olarak RAM'de Heap'te tutulurlar.

- Dizi içerisinde her türlü (değer ya da referans) değer konulabilir. Ama bir dizi tek bir türde değer alabilir.

Dizinin türü integer ise değerleri de integer olmalıdır.

- Dizi içerisine konulan değerler işlevsel olarak aynı mahiyette olmalıdır. Örneğin yaş dizisine maaş bilgisi verilmemelidir. Aynı türde olması mantığını bozmamalı.

- Diziler içerisinde eleman eklendikçe bunları sıralı/düzenli bir şekilde depolayacaktır.

- Dizilerde eklenen elemanlar index ismini verdiğimiz numaralarla ardışık bir şekilde numaralandırılırlar.

- index no: Her bir elemana verilen ardışık sayı. 0'dan başlar n-1'e kadar gider.

Prototip:

type a; ---> değişken

type[]a ---> Dizi

Bir değişken tanımlanırken türünün yanına [] koyarsak o değişken o türde bir dizi değişkeni olacaktır. Bu operatörüre

-indexer- ismi verilir.

type [] name = new type [...dizinin alacağı eleman sayısı...];

- İleride OOP'de göreceğimiz new, bir dizi nesnesi oluşturmamızı sağlar.

int[] yaslar = new int [5];

bool[] x = new int [7];

byte[] sayilar = new byte [n];

- Dizi tanımlama sürecinde eleman sayısı mecburi girilmek zorundadır. Yani dizide çalışılacak değer adedi başta bildirilmelidir.

- Dizilerde tanımlama yaparken eleman sayısının bildirilmesi zorunluluğu bir sınırlılıktır.

int[] yaslar = new int [5];

STACK HEAP

yaslar obj(int[] array)

//Stack'deki yaslar isminin Heap'deki bir nesneyi referans etmesi.

- Bir dizi tanımlaması yağıldığı an bellekte o diziyi kullanmasakta verilen eleman sayısı kadar alan tahsisinde bulunulur.

Bu bir sınırlılıktır. Ekstradan maliyete sebep olur. Kapatıldıktan sonra o alan geri alınır fakat çalışma sürecinde maliyetlidir.

- Diziler alan tahsisi yapıldığında ilgili alanlara türüne uygun default değerleri atarlar.

int[] yaslar = new int [5]; // burada 5 değerde 0 atanacaktır. Ayrıca index numaraları da 0'dan başlayarak atanacaktır.

**Tanımlanmış Diziye Değer Atama**

- İndex numarası dizilere sistem tarafından otomatik verilen ve kimlik mahiyetinde kullanabileceğimiz bir numaradır.

- Her bir elemana karşılık gelen unique değerlerdir. Böyle olması demek her bir elemana istediğimiz zaman erişip değer atayabilmemizi veya var olan değeri okuyabilmemizi sağlar.

int[]yaslar = new int[7];

yaslar[3]; //yaslar dizisindeki 3 index'ine karşılık gelen değeri getir.

- Diziye değer koyarken;

yaslar[3] = 10; // index 3'e 10 değeri atandı.

yaslar[2] = 5;

yaslar[0] = 5;

yaslar[5] = 123;

- Dizinin içerisindeki elemanlara değer atarken değişken davranışı gözlenir. Dolayısıyla herhangi bir elemana atanan en sonuncu değer geçerlidir.

- Dizide aralık dışındaki bir index ile işlem yaparsak Runtime hatası alırız.

yaslar[15] = 123; //yapılamaz. Array'in boyutu 7'dir. Dizinin sınırı neyse onunla çalışmamız lazım.

yaslar[7] = 123; //index no: 0-1-2-3-4-5-6 olduğundan 7. index yok. Yine hata verir.

- Dizilerde değer ataması yaparken sıralamayı göz önünde tutmak zorunda değiliz.

- Dizilerde eleman sayısının başta belirlenmesi, ihtiyaca göre daha fazla genişleletemeceğinden dolayı bir sınırlılıktır.

**Tanımlanmış Diziden Değer Okuma**

int[]yaslar = new int[7];

yaslar[3] = 10;

CW(yaslar[3]); //diyerek ekrana 10 yazdırabiliriz.

CW(yaslar[6]); //ekrana sıfır default değeri yazdırılır.

- Dizilerden değer okurken değer aralığının aşılmamasına dikkat edilmelidir.

Dizi İçerisinde Döngüyle Dönme

string[] personeller = new string[10];

personeller[0] = "Ali";

personeller[1] = "Vali";

personeller[2] = "Mali";

personeller[3] = "Dali";

personeller[4] = "Veli";

personeller[5] = "MeAli";

personeller[6] = "Zalim";

personeller[7] = "Faruk";

personeller[8] = "Tarçın";

personeller[9] = "Apaçi";

Amele'us Yöntemi;

Console.WriteLine(personeller[0]);

Console.WriteLine(personeller[1]);

Console.WriteLine(personeller[2]);

.

.

.

ile son index'e kadar yazdırabiliriz. Ama biz Amele'us'cu değiliz.

- Diziler genellikle "döngülerle" birlikte algoritmalarda kullanılmaktadır.

for (int i = 0; i < 10; i++)

{

Console.WriteLine(personeller[i]);

}

ile tüm değerleri ekrana yazabiliriz.

ya da do while ile:

int i = 0;

do

{

System.Console.WriteLine(personeller[i++]);

} while (i < 10);

- Dizi içerisinde dönecek olan döngü kombinasyonunda dizinin eleman sayısını manuel bir şekilde kullanmamalı. Bu sayısal değeri dizinin kendisinden almalıyız. Bunun için Length, dizinin kaç elemanlı olduğunu int olarak döndüren özellik,'e ihtiyaç vardır.

for (int i = 0; i < personeller.Length; i++)

{

Console.WriteLine(personeller[i]);

}

Not: Dizilerde elemanlara değer atanırken indexer operatörü çokça kullanılmaktaıdır. Bu durum da bir sınırlılıktır. Tüm bu sınırlamaları koleksiyonlar konusunda aşacağız.

**Dizi Tanımlama Varyasyonları**

1)

int[] yaslar = new int [3]; //Temel tanımlamadır.

yaslar[2] = 123; //gibi değer de atayabiliriz.

2)

int[] yaslar = new int [3];

int[] yaslar = { 30, 25 , 42 ,52 }; //gibi bir değer atama yöntemi de vardır.

ya da tek satırda hem tanımlama hem atama yapılabilir.

string[] isimler = { "Ali", "Gullit", "Zubizaretta", "Tanjantberk" }; //ne kadar değer verirsek o kadarlık dizi oluşturur

3)

string [] isimler = new string[] { "Ali", "Gullit", "Zubizaretta", "Tanjantberk" };//bu da bir varyasyondur.

4)

string [] isimler = new string[3] { "Ali", "Gullit", "Zubizaretta" }; //new string[3] belirttiğimiz an tam 3 tane isim girmek zorundayız. Ne eksik ne fazla.

5)

int[] sayilar = new[] {3, 5, 7};

yapıldığı gibi,

var sayilar = new[] {2, 3, 5, 7};

var sayilar2 = new[] {"2", "3", "5", "7", "Alican"}; //gibi türlü türlü varyasyonlar vardır.

**Array Sınıfı Nedir? Ne Amaçla Kullanılır?**

- Dizi olarak tanımlanan değişkenler Array sınıfından türetilmektedirler.

- Dolayısıyla dizilerde Array sınıfından gelen belirli metodlar ve özellikler mevcuttur.

1)Bu format genellikle algoritmalarda tercih edilir. Çünkü -indexer- algoritmalarda çokça kullanılır.

int[] yaslar = new int[3]; //Dizi eğer ki kendi türünde tutuluyorsa/referns ediliyorsa indexer operatörü kullanılabilir. Bu şekilde çalışıldığında ilgili diziye verisel müdaheleler operatif gerçekleştirilir.

2) Genellikle elimizdeki dizinin üzerinde işlem yaparken tercih edilir. Dizi hakkında bilgi edilirken vs.

Array yaslar = new int[3]; //Eğer Array türünde referans ediliyorsa indexer operatörü kullanılamaz. Bu şekilde ise fonksiyonel çözümler getirmektedir. Veri ekleme/çağırma fonksiyon üzerinden gerçekleşir.

**Array Türünden Dizilere Değer Atama - Okuma**

**Değer Atama:**

1.Yöntem

int[] dizi = new int [3];

dizi[0] = 30;

dizi[1] = 20;

dizi[2] = 40;

Array dizi2 = dizi; //1. formatta oluşturulan diziyi Array dizi'sinde tutabiliriz.

2. Yöntem

Array dizi = new int[] {3,5,7,9}

Array dizi = new int[4] {3,5,7,9}

ya da

Array dizi = new[] {3,5,7,9} ile dizi tanımlama yapabiliriz.

Not: Array dizi = {3,5,7,9} tarzı bir tanımlama yapılamaz.

3. Yöntem

Array dizi = new int[3];

dizi.SetValue(30,0); //30 değerini 0. index'e ata.

dizi.SetValue(20,1); //20 değerini 1. index'e ata.

dizi.SetValue(40,2); //40 değerini 2. index'e ata.

**Değer Okuma:**

object value = dizi.GetValue(1);

CW(value);

- GetValue, Array'in içerisinde hangi türden dizi olduğuyla ilgilenmez. Onun için ilgili değer ne ise object'e boxing eder ve o şekilde bize gönderir. Biz de gerektiği

taktirde bunu unboxing yapıp kendi türünde/değerinde kullanırız.

**Array Sınıfı Metodları**

**Clear Motodu**

- Dizi içerisindeki tüm elemanlara, dizinin türüne uygun default değerleri atayan bir fonksiyondur. Dizi içerisindeki tüm elemanları siler tanımı yanlıştır.

Array isimler = new[] { "Hilmi", "Hüseyin", "Şuayip" , "Farukgül" , "Hakansu" };

Array.Clear(isimler, 0 , isimler.Length); //0. index'ten Length kadar o kadar eleman kadar Clear'la.

Array.Clear(isimler, 1 , 3); //1.index'ten başla 3. index'e kadar. 1-2-3'ü Clear'la.

**Copy Metodu**

- Elimizdeki bir dizinin verilerini bir başka diziye kopyalamamızı sağlayan bir fonksiyondur.

Array isimler = new[] { "Hilmi", "Hüseyin", "Şuayip" , "Farukgül" , "Hakansu" }; //aşağıdaki örneklerde bu diziyi kullanacağız.

string[] isimler2 = new string[isimler.Length];

Array.Copy(isimler, isimler2, 5);

for(int i = 0; i < isimler2.Length; i++)

Console.WriteLine(isimler2[i]); //isimler'in tüm içeriğini isimler2'ye kopyaladık.

ya da belirli bir aralıkta da çalışabiliriz.

Array.Copya(isimler, 2, isimler2, 1, 3); //kaynak olan isimler'den, 2. index'ten başlayarak, isimler2'ye, kopyalanacak yerin başlangıç index'i olan 1'den itibaren, 3 tane elemanı ata. Yani 1-2-3. index'leri

for(int i = 0; i < isimler2.Length; i++)

Console.WriteLine(isimler2[i]);

Çıktı:

(boşluk-null)

Suayip

Farukgül

Hakansu

(boşluk-null)

olur.

**IndexOf Metodu**

- Dizi içerisinde bir elemanın var olup olmadığını sorgulayabildiğimiz fonksiyondur.

int index = Array.IndexOf(isimler, "Rıfkı"); //eğer dizimizde Rıfkı değeri varsa bize integer olarak Rıfkı'nın index numarasını döndürür.

if(index != -1)

{

Console.WriteLine("Var.");

}

//yoksa -1 değerini döndürür.

ya da belirtilen yer aralığında arama yapabiliriz.

int index = Array.IndexOf(isimler, "Rıfkı", 0, 3); //0. index'ten başlayıp 3 elemanda Rıfkı'yı ara. Yani 0-1-2'de ara.

if(index != -1)

{

Console.WriteLine("Var.");

}

**Reverse Metodu**

- Elimizdeki dizinin elemanlarını tersine sıralayan bir fonksiyondur.

Array isimler = new[] { "Hilmi", "Hüseyin", "Şuayip" , "Farukgül" , "Hakansu" };

for(int i = 0; i < isimler.Length; i++)

Console.WriteLine(isimler.GetValue(i));

Array.Reverse(isimler);

Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

for(int i = 0; i < isimler.Length; i++)

Console.WriteLine(isimler.GetValue(i));

Çıktı:

Hilmi

Hüseyin

Suayip

Farukgül

Hakansu

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Hakansu

Farukgül

Suayip

Hüseyin

Hilmi

ya da verilen index'ten sonraki (o index dahil) n elemanı da Reverse'leyebiliriz.

Array.Reverse(isimler, 0, 3); //0-1-2'yi reverse et. Diğerlerine dokunma.

**Sort Metodu**

Array isimler = new[] { "Hilmi", "Hüseyin", "Şuayip" , "Farukgül" , "Hakansu" };

for(int i = 0; i < isimler.Length; i++)

Console.WriteLine(isimler.GetValue(i));

Array.Sort(isimler); //string ise alfabeye, int ise küçükten büyüğe göre sıralar.

Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

for(int i = 0; i < isimler.Length; i++)

Console.WriteLine(isimler.GetValue(i));

Çıktı:

Hilmi

Hüseyin

Suayip

Farukgül

Hakansu

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Farukgül

Hakansu

Hilmi

Hüseyin

Suayip

**Array Sınıfı IsReadOnly Property'si**

- Dizinin salt okunabilirlik kontrolüne yarar. Dizi hem okunabilir hem yazılabilir ise False döndürür. Sadece okunabilir ise true döndürür.

Array isimler = new[] { "Hilmi", "Hüseyin", "Şuayip" , "Farukgül" , "Hakansu" };

Console.WriteLine(isimler.IsReadOnly);

**Array Sınıfı IsFixedSize Property'si**

- Bir veri kümesinin eleman sayısının sabit olup olmama durumunu öğrenmeye yarar.

Array isimler = new[] { "Hilmi", "Hüseyin", "Şuayip" , "Farukgül" , "Hakansu" };

Console.WriteLine(isimler.IsFixedSize);

//Tüm dizilerde eleman sayısı sabit olduğu için sürekli true dönecektir.

//fakat ArrayList koleksiyonunda false dönmektedir.

**Array Sınıfı Length Property'si**

- Eleman sayısını getirir.

**Array Sınıfı Rank Propertysi**

- İlgili dizinin derece sayısını döndürür.

Array isimler = new[] { "Hilmi", "Hüseyin", "Şuayip" , "Farukgül" , "Hakansu" };

Console.WriteLine(isimler.Rank); //1 değerini döndürür çünkü 1 dereceli dizilerde çalıştık.

int[,,] x = new int [3,4,5]; // Rank'i 3 olan bir dizi

**Array Sınıfı - CreateInstance Metodu İle Dizi Tanımlama**

int[] yaslar = new int[3];

- Normalde yukarıdaki gibi yapılan dizi tanımlaması esasında arkaplanda Array sınıfının CreateInstance metodunu kullanmaktadır.

Bizler de bu metodu kullanarak fonksiyonel diziler oluşturabilmekteyiz.

Array yaslar2 = Array.CreateInstance(typeof(int), 3); //ile de 3 boyutlu integer bir dizi oluşturabiliriz.

**Array Sınıfı - CreateInstance Metodu İle Çok Boyutlu Dizi Tanımlama**

Array dizi = Array.CreateInstance(typeof(int), 3 , 5 , 2 , 6 ); // Dizinin derecesi 4, her bir derecenin elemanları sırasıyla 3-5-2-6'dır.

Console.WriteLine(isimler.Rank); //4 yazdırır.

İleride daha detaylı.

Örnek:

Dizinin elemanlarını kullanıcıdan alma ve ekrana yazdırma.

using System;

namespace OrnekProje

{

class Program

{

static void Main(string[] args)

{

int[] sayilar = new int[4];

for (int i = 0; i < sayilar.Length; i++)

{

System.Console.Write("Sayılar dizisinin " + (i) + ". index değerini giriniz: ");

sayilar[i] = Convert.ToInt32(Console.ReadLine());

}

System.Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

for (int i = 0; i < sayilar.Length; i++)

{

System.Console.WriteLine(sayilar[i]);

}

}

}

}

Çıktı:

Sayilar dizisinin 0. index degerini giriniz: 40

Sayilar dizisinin 1. index degerini giriniz: 50

Sayilar dizisinin 2. index degerini giriniz: 40

Sayilar dizisinin 3. index degerini giriniz: 06

\*\*\*\*\*\*\*\*\*\*\*\*\*\*

40

50

40

6

Örnek:

Kullanıcıdan girdiği sayı dizinin içinde var mı diye kontrol eden varsa kaç tane olduğunu söyleyen yoksa da yoktur diye uyaran program.

using System;

namespace OrnekProje

{

class Program

{

static void Main(string[] args)

{

int[] sayilar = {10,10,20,30,20,30,40,50,70,55,65,70,55,65,5,20,35,5,20,34,65,90};

int adet = 0;

int sayi;

System.Console.Write("Lütfen bir sayı giriniz: ");

sayi = Convert.ToInt32(Console.ReadLine());

for (int i = 0; i < sayilar.Length; i++)

{

if( sayi == sayilar[i])

{

adet++;

}

}

if (adet == 0)

{

System.Console.WriteLine("Girdiğiniz değer bulunamadı.");

}

else

{

System.Console.WriteLine("Girdiğiniz değer dizide bulundu.");

System.Console.WriteLine("Adet: " + adet);

}

}

}

}

Çıktı:

Lütfen bir sayi giriniz: 10

Girdiginiz deger dizide bulundu.

Adet: 2

Çıktı:

Lütfen bir sayi giriniz: 300

Girdiginiz deger bulunamadi.

Örnek:

Kullanıcıdan 10 tane int değer alan ve ;

1- Kaç adet tek, çift eleman olduğunu,

2- Çift değerlein toplamını, tek değerlerin toplamını bulan.

3- Son olarak da çift değerlerin toplamı, tek değerlerden büyükse çift değerlerin toplamı büyüktür yazdıran tersi geçerli ise tek değerlerin toplamı büyüktür yazdıran program.

using System;

namespace OrnekProje

{

class Program

{

static void Main(string[] args)

{

int[] sayilar = new int [10];

int tekAdetToplam = 0;

int ciftAdetToplam = 0;

int tekSayilarinToplami = 0;

int ciftSayilarinToplami = 0;

int kullaniciDeger;

for (int i = 0; i < sayilar.Length; i++)

{

System.Console.Write((i) + ". index değerini giriniz: ");

kullaniciDeger = int.Parse(Console.ReadLine());

sayilar [i] = kullaniciDeger; //dizinin içini kullanıcıdan aldığımız değerler ile doldurduk.

}

for (int j = 0; j < sayilar.Length; j++)

{

if(sayilar[j] % 2 == 0)

{

ciftAdetToplam++;

ciftSayilarinToplami += sayilar[j];

}

else

{

tekAdetToplam++;

tekSayilarinToplami += sayilar[j];

}

}

if(ciftSayilarinToplami > tekSayilarinToplami)

System.Console.WriteLine("Çift sayıların toplamı tek sayıların toplamından büyüktür.");

else

System.Console.WriteLine("Tek sayıların toplamı çift sayılarından büyüktür.");

System.Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

System.Console.WriteLine("Çift sayıların Toplamı: " + ciftSayilarinToplami);

System.Console.WriteLine("Çift sayıların adedi: " + ciftAdetToplam);

System.Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

System.Console.WriteLine("Tek sayıların Toplamı: " + tekSayilarinToplami);

System.Console.WriteLine("Tek sayıların adedi: " + tekAdetToplam);

}

}

}

Çıktı:

0. index degerini giriniz: 10

1. index degerini giriniz: 20

2. index degerini giriniz: 50

3. index degerini giriniz: 40

4. index degerini giriniz: 10

5. index degerini giriniz: 1

6. index degerini giriniz: 2

7. index degerini giriniz: 5

8. index degerini giriniz: 8

9. index degerini giriniz: 9

Çift sayilarin toplami tek sayilarin toplamindan büyüktür.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Çift sayilarin Toplami: 140

Çift sayilarin adedi: 7

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Tek sayilarin Toplami: 15

Tek sayilarin adedi: 3

**DERS-11: Ranges and Indices**

- C# 8.0 ile veri kaynakları üzerinde gerekli manipülasyonu sağlayabilmek ve bunun yanında kaynak içerisindeki

tüm veriler üzerinde yapılan genel sorgulamalar ve algoritmalardan kaçınmak, yani direkt olarak hedef veri/ler

odaklı çalışabilmek için yeni tipler ile operatörler geliştirilmiş bulunmaktadır.

**System.Index**

- Dizi ve koleksiyon yapılarındaki index kavramının tip olarak belirlenmiş halidir.

- Temelde index değerini bir tür ile tutmakla beraber ^ operatörüyle birlikte daha fazla anlam ifade etmekte ve dizinin index değerlerini tersine ifade edecek şekilde sorumluluk yüklenmektedir.

Index i = 3;

// 0 1 2 (3) 'ü alır. 3 değerine sahip olan bir index'e karşılık gelecektir. Normal gidişatta 0'dan başlar n-1'e doğru gider.

Index i = ^3; // (3) 2 1 ile index'i tersinden almaya başladığımızda saymaya 0'dan değil 1'den başlarız. n'e doğru gideriz.

Örnek:

string[] isimler = { "Ali", "Veli", "Rıfkı", "Hatice", "Ayşe" };

----0------1-------2-------3-->

Index index = 3; //Hatice'yi gösterir.

Fakat,

string[] isimler = { "Ali", "Veli", "Rıfkı", "Hatice", "Ayşe" };

<-3--------2---------1

Index index = ^3; //Rıfkı'yı gösterir.

Not: Indexer[] operatörü içerisinde tam sayı verilebildiği gibi Index türü de verilebilir.

sayilar[3];

sayilar[sayi]

sayilar[index];

Örnek:

int[] sayilar = {3, 5, 7, 9, 1, 2, 31, 321 , 534, 5643, 457, 56, 78 };

Index index = 5; //2'yi soldan yakalamak istiyorsak..

Console.WriteLine(sayilar[index]);

Ekrana 2'yi yazdıracaktır.

//ya da sağdan yakalamak istersek;

Index index = ^8;

Console.WriteLine(sayilar[index]);

Ekrana 2'yi yazdıracaktır.

**System.Range**

- Veri kümelerinde hangi değerler ile çalışacağımızı belirleyebilmek için index üzerinden aralık vermemizi ve bunu .. operatörü ile gerçekleştirmemizi sağlayan yapılanmadır.

string[] isimler = { "Hüseyin", "Hilmi", "Rıfkı", "Hatice", "Ayşe", "Mahmut", "Kazım", "Okan", "Emre", "Ali", "Naci", "Hüsnü", "Kamil" };

----0----------1--------2--------3-> <-7--------6------5-------4------3--------2---------1------

Range range = 3..7; //3 index no iken 7 sıra no'dur.

Not: .. operatörü hedeflenen aralığı bizlere Range türünde geri getirecektir. Böylece biz ilgili aralığı biz diziymiş gibi kullanabileceğiz.

Örnek:

string[] isimler = { "Hüseyin", "Hilmi", "Rıfkı", "Hatice", "Ayşe", "Mahmut", "Kazım", "Okan", "Emre", "Ali", "Naci", "Hüsnü", "Kamil" };

----0----------1--------2--------3-> <---7-------6---------5--------4-----3-----------2---------1-----

Range range = 3..^6;

//sağdaki sayıya tersinden gelirken operatörü ^ kullanıllırsa sağdaki sayımdan gelen ifade alınmayacak ondan bir önceki alınacaktır. Yani Okan değil Kazım alınacak ve aralık "Hatice", "Ayşe", "Mahmut", "Kazım" olacaktır.

string[] isimler = { "Hüseyin", "Hilmi", "Rıfkı", "Hatice", "Ayşe", "Mahmut", "Kazım", "Okan", "Emre", "Ali", "Naci", "Hüsnü", "Kamil" };

13--------12-------11--------10--------9--------8-------------7-------(6)------5-------(4)------3--------2----------1

Range range = ^6..^3; //^6 solda olduğu için bir öncesine kaymaz.

Aralığımız "Okan", "Emre", "Ali" olacaktır.

Not: Indexer[] operatörü içerisine tam sayı veya Index türü verilebildiği gibi Range türü de verilebilir.

sayilar[range]

int[] sayilar = {3, 5, 7, 9, 1, 2, 31, 321 , 534, 5643, 457, 56, 78 };

Range range = 5..10; //5. ile 10. index arasını aldık.

var sayilar2 = sayilar[range]; //ilgili dizinin ayrılan alanını ayrı bir dizi olarak getirdi.

sayilar2[0] = 100; //yeni dizi üstünde istediğimiz değişikliği yapabiliriz. Bu değişiklikler ilk sayilar dizisini etkilemez.

Not:

Range range = ..; -> Tüm diziye karşılık gelir.

var sayilar2 = sayilar[range]; //ilgili diziyi kopyalacaktır.

Örnek:

int[] sayilar = {3, 5, 7, 9, 1, 2, 31, 321, 534, 5643, 457, 56, 78 };

Range range = 5..^3;

var sayilar2 = sayilar[range]; //sayilar 2'ye 2, 31, ,321 , 534, 5643'ü alır.

Range range = ^8..^3;

var sayilar2 = sayilar[range]; //sayilar 2'ye 2, 31, ,321 , 534, 5643'ü alır.

Örnek:

nt[] sayilar = {3, 5, 7, 9, 1, 2, 31, 321, 534, 5643, 457, 56, 78 };

Index i1 = 5, i2 = 10;

Range range = i1..i2; //gibi de kullanılabilir.

Örnek:

string isim = isimler[isimler.Length - 1];

string isim = isimler[^1];

iki kodda dizinin en sondaki elemanını vermeye yarayacaktır.

Not: ^ operatörünü kullanabilmek için index türünde çalışılması gerekmektedir. Çünkü bu operatör geriye index türü döndürür.

int i = ^3; //Yanlış

Index i = ^3; //Doğru. Sağdan başlar 1-2-3 diye sayar.

**DERS-12: ÇOK BOYUTLU DİZİLER**

- Bugüne kadar tek boyutlu diziler kullandık.

- Çok boyutlu diziler oyun programlamada yahut yüksek istatiksel çalışmalarda kullanılan bir yapıdır.

**Çok Boyutlu Dizi Tanımlama**

type[] -> tek boyutlu diziler

type[,] -> iki boyutlu bir dizi(çift)

type[,,,] -> dört boyutlu dizi

- Çok boyutlu dizilerde indexer içerisinde virgül , ile tanımlama yapılmaktadır.

- Virgül sayısı + 1 değeri dizinin derecesini vermektedir.

int[] sayilar = new int[3];

int [,] sayilar = new int[3,5]; //2 boyutlu diziler x ve y koordinatları ile tahayyül edebileceğimiz dizilerdir.

int[,,,] = new int [3, 4, 5, 6]; //2 dereceliden fazla olan dizileri günlük hayatta tahayyül edebilmek pek mümkün değil

//Fakat programatik olarak inşa edebilmekteyiz.

**Tanımlanmış Çok Boyutlu Diziye Değer Atama**

int[,] sayilar = new int [3,4]; //x'te 0,1,2 ve y'de 0,1,2,3 olmak üzere index'leneceklerdir.

**0 1 2**

**0** (2,0)

**1** (0,1)

**2** (1,2)

**3**

sayilar [1,2] = 5; //x'i 1, y'si 2 olan alana 5 değerini yaz.

sayilar [2,0] = 15;

Çok dereceli dizileri örneklendirelim:

int [,,,] sayilar = new int [2,3,4,5];

sayilar[0,0,0,1] = 15;

sayilar[0,0,0,2] = 16;

sayilar[0,0,0,3] = 17;

sayilar[0,0,0,4] = 18;

sayilar[0,0,0,5] = 19; //dördüncü derecenin tüm değerleri atandı.

sayilar[1,0,0,0] = 115; //birincinin en sonuna ata.

Örnek:

int[,,] sayilar = new int [2,3,4];

sayilar [0,0,1] = 15; //görsel olarak hayal etmek zordur.

**Çok Boyutlu Dizilerde Değer Atama - Farklı Varyasyonlar**

Örnek:

int sayilar[,] sayilar = {

{ 3,5,7 }, //birinci derecenin kolonlarını belirledik.

{ 5,6,7 }

{ 15, 36, 57 },

{ 25, 46, 67 }, //y derecesi 4 satır oldu toplamda. 3'e 4'lük bir dizi oluştu.

};

**0 1 2**

**0** 3 5 7

**1** 5 6 7

**2** 15 36 57

**3** 25 46 67

**Çok Boyutlu Dizilerden Değer Okuma**

int [,] sayilar = new int [4,3];

sayilar [0,0] = 3;

sayilar [0,1] = 5;

sayilar [0,2] = 7;

sayilar [1,0] = 5;

sayilar [1,1] = 6;

sayilar [1,2] = 7;

sayilar [3,2] = 67;

Console.WriteLine(sayilar[3,2]);

Console.WriteLine(sayilar[0,0]);

Çıktı:

67

3

**Dizinin Derecesini Öğrenme(Rank Özelliği)**

Console.WriteLine(sayilar.Rank); //ilgili dizinin Rank değerini bize getirecektir. Yukarıdaki dizinin Rank değeri 2'dir.

**Çok Boyutlu Dizilerin Eleman Sayısını Hesaplama**

int[,,] sayilar = new int[2,2,4]; // toplam eleman sayısını bulmak için tüm boyutların eleman sayılarını çarpmak yeterlidir. 2\*2\*4 = 16'dır. Dizinin Length'i 16'dır.

//bu dizi Rank'i 3 olan 16 elemanlı bir dizidir.

Console.WriteLine(sayilar.Length); //16

Sadece bir boyutun Length'ini öğrenmek için ise;

Console.WriteLine(sayilar.GetLength(0));

Console.WriteLine(sayilar.GetLength(1));

Console.WriteLine(sayilar.GetLength(2));

Çıktı:

2

2

4

**Çok Boyutlu Dizilerdeki Verileri İç İçe Döngülerle Ekrana Yazdırma**

Örnek: for (int i = 0; i < sayilar.GetLength(0); i++)

{

for (int j = 0; j < sayilar.GetLength(1); j++)

{

for (int h = 0; h < sayilar.GetLength(2); h++)

{

Console.Write(sayilar[i,j,h] + " ");

}

Console.WriteLine("");

}

}

Çıktı:

0 0 0 0

0 0 0 0

0 0 0 0

0 0 0 0

- Çıktıya çok takılmadan her bir boyuta nasıl GetLength ile girilip, en içteki loop'ta e sonuncu boyutta tam tur attığımızı görmek lazım.

**Düzensiz Diziler Nedir?**

- Düzensiz diziler her bir elemanı kendi içerisinde farklı bir dizi barındıran dizilerdir.

- Çok boyutlu dizilerden tek farkı, çok boyutlu dizilerin sütun sayılarının sabit, düzensiz dizilerin ise sütun sayılarının değişken olmasıdır.

type[]; //type türünde bir dizi tanımladık.

type[][] //type dizisi türünde değerler tutacak bir dizi tanımladık.

int[][] sayilar = new int[3][]; //dizi içerisinde integer türünden diziler barındıran bir dizi tanımladık.

**Değer Atama:**

sayilar[0] = new int[3] {3,5,7};

sayilar[1] = new int[5] {3,5,7, 523, 01};

sayilar[2] = new int[10] {3,5,7,234,23,4,234,234,123};

**Değer Okuma:**

Console.WriteLine(sayilar[0][0]);

sayilar[0][0] = 13;

Console.WriteLine(sayilar[0][0]);

Çıktı:

3

13

**Eleman Sayısı Öğrenme:**

Console.WriteLine(sayilar.Length); //düzensiz dizinin eleman sayısını getirir. Bu bize total eleman sayısını vermez. Çok boyutlu dizilerde olduğu gibi düzensiz diziler

//başlı başına bir dizi yapısı değildir. Normal dizi yapılanmasıdır. Lakin içerisinde dizi barındıran bir dizidir. Haliyle eleman sayısını döndürür.

Ekrana 3 yazdırır. Ana dizinin eleman sayısıdır.

//İçteki dizilerin eleman sayılarını totalde elde edebilmek için her birini toplamamız gerekir.

sayilar[0] = new int[3] {3,5,7};

sayilar[1] = new int[5] {3,5,7, 523, 01};

sayilar[2] = new int[10] {3,5,7,234,23,4,234,234,123};

3+5+10 = 18 tane elamanı vardır.

Console.WriteLine(sayilar[0].Length + sayilar[1].Length + sayilar[2].Length);

Ekrana 18 yazdırır. Toplam eleman sayısıdır.

**Düzensiz Dizilerde İç İçe Döngülerle Çalışma**

static void Main(string[] args)

{

int[][] sayilar = new int[3][];

sayilar[0] = new int[3] {3,5,7};

sayilar[1] = new int[5] {3,5,7, 523, 01};

sayilar[2] = new int[10] {3,5,7,234,23,4,234,234,123,212};

for (int i = 0; i < sayilar.Length; i++)

{

for (int j = 0; j < sayilar[i].Length; j++)

{

Console.Write(sayilar[i][j] + " - ");

}

Console.WriteLine("");

}

}

Çıktı:

3 - 5 - 7 -

3 - 5 - 7 - 523 - 1 -

3 - 5 - 7 - 234 - 23 - 4 - 234 - 234 - 123 - 212 -

**DERS-13: String Türü**

- Referans türlü olupta programlama dilinde bir keyword'le karşılanabilen/keyword barındıran tek türdür.

int a = 5;

string b = "Hilmi";

STACK HEAP

(int) a = 5 "Hilmi"

(string)b

Değişken kısmı Stack'te değer kısmı HEAP'te tutulacaktır. b, "Hilmi"yi referans eder.

**Null - Empty Durumları, Farkları**

- Bir değişken/nullable/referans eğer ki null alıyorsa bu durum ilgili değişkenin herhangi bir alanı tahsis etmediği

anlamına gelir.

STACK HEAP

x //x null demek bir değer karşılığı yok demek.

- Bir değişken/nullable/referans eğer ki empty ise bu değişkenin değeri yok anlamına gelir. İçi boş olan ama alan tahsisi yapılmış bir değişkendir.

- Değer türlü değişkenler null alamazlar. Sadece referans türler null alabilir. Değer türlü değişkenlerin null alabilmesi için nullable(?) olmaları gerekir.

int? a = null;

- Tüm değerlere empty atanabilir. Default değerlerin olduğu durumlar empty olarak geçerlidir.

int a = 0;

bool b = false;

int[] x = new int[55];

- Empty dendiğinde aklınıza string'e "" değerinin verilmesi gelsin.

STACK HEAP

x---------------->

//karşılığı var ama empty. Boş. Bellekte yer kaplayacak ve alan tahsisinde bulunmuş olacaktır.

string a = "";

string a2 = string.Empty;

- İki varyasyonda empty işlemi yapılabilir.

- Null olan bir değer üzerinde işlem yapmaya çalıştığımızda runtime hatası meydana gelir. Empty olan bir değer üzerinde işlem gerçekleştirilebilir.

**String İfadelerde IsNullOrEmpty Kontrolü**

- Elimizdeki string ifadelerin işleme tabi tutulmasından önce kesinlikle kontrol etmemiz gerekli.

string x = "";

//string x = string.Empty;

if(x != string.Empty && x is not null)

{

… //ilgili operasyonlar...

}

ya da

- IsNullOrEmpty fonksiyonu elimizdeki string ifadenin null yahut empty olup olmasa durumları hakkında bir check yapar ve geriye bool türde sonuç döner.

if(string.IsNullOrEmpty(x)) // eğer ki değer null ya da empty ise true değilse false dönecektir.

{

…

}

fakat genel kullanımda,

if(!string.IsNullOrEmpty(x)) // değilse ilgili operasyonları yap şekline kullanılır.

{

…

}

String İfadelerde IsNullOrWhiteSpace Kontrolü

string x = " "; //boşluk barındıran değer

if(string.IsNullOrWhiteSpace(x))

{

…

}

IsNullOrWhiteSpace fonksiyonu elimizdeki string ifadenin null, empty yahut boşluk karakterlerinden ibaret olma durumunda geriye bool true değerini döndürür.

if(!string.IsNullOrWhiteSpace(x)) //genelde bu şekilde kullanılır.

- String ifadeler aslında bir char dizisidir. Yazılım açısından string ifade yoktur. Esasında karakterlerin bir araya gelmiş hali vardır. Karakterleri de bir araya getirebilecek yegane şey de bir dizidir.

- String ifadeler yazılımsal açıdan bilgisayarda bir char dizisi olarak tarif edilmekte ve o şekilde tutulmaktadırlar.

string metin = "You'll never walk alone!";

- String ifadeler özünde bir dizi olmasından dolayı referans türlü değişkenlerdir. Çünkü diziler nesnedirler, referans türlüdürler ve HEAP'te tutulurlar.

- String ifadeler char dizi olduklarından dolayı yapısal olarak her bir karakter baştan sona otomatik index'lenmektedir. Dolaysıyla string bir ifade üzerinde bizler indexer operatörünü de kullanabilmekteyiz.

metin[3]; //You' 0 1 2 3. index tırnak işaretine denk gelecektir.

CW(metin[3]); // ' yazar.

CW(metin.Length); // Length de kullanılabilir.

Array array = metin; //string özünde bir char dizisi olabilir ama yapısal olarak yine de string olduğu için Array referansına atanamaz. Array ile karşılanamaz.

**Döngülerle String Metin İçerisindeki Her Bir Karaktere Ulaşma**

string metin = "Aliiiii alidesideroo";

for (int i = 0; i < metin.Lenght; i++)

{

if(metin[i] == 'l');

CW(i);

}

İlgili metindeki l karakterlerinin bulunduğu index numaralarını elde ederiz.

string metin = "Aliiiii alidesideroo";

int i = 0;

do

{

if(metin[i] == 'l')

CW(i);

i++;

}while (i < metin.Length);

**String İfadelerde + Operatörü Kullanımı**

+ Operatörü kullanılabilmektedir. İki string ifade arasında yanyana birleştirme işlemini görür.

string a = "merhaba", b = "dünya";

Console.WriteLine(a+b); //Ekrana merhabadünya yazılır.

- Bir string ifade herhangi bir tür + operatörü ile işleme tabi tutulabilir.

- + operatörü string bir ifadeye herhangi bir türdeki ifadeleri işleme tabii tutarken object + string davranışı sergilemektedir.

string a = "merhaba";

int a2 = 5;

a + a2 -> string left, object right ise string döndürür. Tam tersi de aynı. String'in yanında ne +'larsan object+string davranışı sergiler. Geriye string döndürür.

- Dolayısıyla herhangi bir ifadeyi string'e dönüştürebilmek için o ifadeyi + "" ile işleme tutmak yeterli olabilir.

Console.WriteLine(5 + 7 + 20 + "Ahmet"); //Ekrana 32Ahmet yazdırır. Önce aritmetik işlemi yapar.

Fakat,

Console.WriteLine("Ahmet" + 3 + 5 + 22); // Ekrana Ahmet3522 yazdırır.

**String Formatlandırma**

Örnek:

String Formatlandırma örneği.

string isim = "Ali", soyisim = "Gümüş", tcNo = "12345678910";

int yas = 28;

bool medeniHal = true;

Console.WriteLine(" TC No: ................ olan ...... ....... şahsın bilgileri | Yaş : .. | Medeni Hal : .. ");

noktalı yerlere değerleri uygun şekilde yerleştirmemiz lazım.

**+ Operatörü ile Formatlandırma**

Console.WriteLine(" TC No: " + tcNo + " olan " + isim + " " + soyisim + " Adlı şahsın bilgileri | " + "Yaş : " + yas + " | Medeni Hal : " + (medeniHal ? "Evli" : "Bekar"));

Not: String + ile birleştirme yapılırken ternary operatörünü paranteze almayı unutmamak lazım.

Çıktı:

TC No: ................ olan ...... ....... sahsin bilgileri | Yas : .. | Medeni Hal : ..

TC No: 12345678910 olan Ali Gümüs Adli sahsin bilgileri | Yas : 28 | Medeni Hal : Evli

- Çok uzun ve maliyetli oldu.

**string.Format Metodu İle Formatlandırma**

- "........{0}.............{1}..............{2}..............", degisken1, degisken2, degisken3

- string.Format fonksiyonu metinsel kalibin içerisindeki indexsel belirlenen noktalara sırasıyla değer göndermemizi sağlayan bir fonksiyondur.

string sonuc = string.Format(" TC No: {0} olan {1} {2} şahsın bilgileri | Yaş : {3} | Medeni Hal : {4} ", tcNo, isim, soyisim, yas, medeniHal ? "Evli" : "Bekar");

Console.WriteLine(sonuc);

**$(String Interpolation) Operatörü İle Formatlandırma (C# 6.0)**

- String interpolation, string ifadeniniçerisinde süslü parantez ile araya girerek programatik bir değişkenin değerini bırakmamızı/eklememizi sağlayan bir operatördür.

$".....{isim}....{soyisim}..."

- Bir string değerin başına $ operatörü koyulursa --> $"..." bu ifadenin içerisinde string interpolation operasyonunun/operatorunun kullanılabilirliği sağlanır. İlgili string içerisindeki süslü parantezler bir interpolation özelliği sergilerler.

Console.WriteLine($" TC No: {tcNo} olan {isim} {soyisim} şahsın bilgileri | Yaş : {yas} | Medeni Hal : {(medeniHal ? "Evli": "Bekar")} ");

- String interpolation yapısal olarak arka planda string.Format fonksiyonuyla şekillenen bir operatördür.

Not: {}'leri de ekranda görünsün istersek ($"......{{alican}}...") çiftli parantez yapabiliriz. İki string interpolation birbirini ezer. Metinsel olarak süslü parantezi kullanabiliriz.

**String Değerlerde (Escape)Kaçış Karakterleri**

"......"

" eylemsel bir karakterdir. String açısından belirli bir operasyonu/eylemi/sorumluluğu üstlenen bir karakterdir. Dolaysıyla böyle bir karakteri metinin içerisinde

salt bir şekilde kullanmamız mümkün değildir.

"........."............" -> ortadaki " gibi bir şekilde string için özel eylemsel mahiyet ifade eden bir karakteri metinsel olarak kullanacaksam bu karakterin o anlık

özel karakter olmadığını ifade etmem gerekmektedir.

- Bunun için escape/kaçış karakterleri kullanılmalıdır.

- String içerisinde kaçış karakteri olarak \ backward slash kullanılmaktadır. String içerisinde özel/operatif karakteri ezen ve bunları metinsel hale getirmemizi sağlayan bir karakterdir.

Console.WriteLine("\"Bu gün hava çok göözeel.\"");

Çıktı:

"Bu gün hava çok göözeel."

- Öyle kafamıza göre de kullanamayız. Yanına illa ki özel bir karakter/eylemsel bir karakter bekler. Aksi halde hata verir.

- Metinsel olarak \ kullanmak istiyorsak eğer ilgili operatörü yine kendisiyle ezmeliyiz.

Console.WriteLine("\\Bu gün hava \\çok göözeel.\\");

Çıktı:

\Bu gün hava \çok göözeel.\

**Kaçış karakterleri**

\o -> Null sonlandırma karakteridir. Genel olarak dosya veya veri kanalının bitişini belirtmek için kullanılır.

\a -> Bip sesi çıkartır.

\b -> Backspace - geri - kendisinden önceki bir karakteri siler

\t -> tab

\r -> satır başı (carriage return)

\n -> bir alt satıra in

\v -> dikey tab

\f -> sayfa ilerleme

\" -> çift tırnak

\' -> tek tırnak

\ -> backlash

char c = '\''; //char ' tek tırnak kullanmak istersek.

**String Türlerde @(Verbatim Strings) Operatörü**

**1.Kullanım**

- Bir değişken yahut metod vs. gibi yapılanma isimlerinin bir keyword'e karşılık gelmesi durumunda derleyici hatası verilir.

İlla ki kullanmak için ölüyorsak @ operatörü kullanabiliriz.

int a = 5; //doğru

int class = 5; -> int @class = 5;

int void = 5; -> int @void = 5;

int namespace = 5; -> int @namespace = 5;

void @void() //metodlarda da kullanılır.

{

}

**2. Kullanım**

string metin = "Hava çok \"güzel\"";

ya da

string metin = @"Hava çok ""güzel""; //@koyduğumuz an ilgili eylemsel karakteri kendisiyle ezmemize yarar.

- Escape karakterlerinin kullanılması gereken durumlarda @ operatörünü kullanarak eylemsel karakterleri kendileriyle ezebilecek özellik kazandırabiliyoruz.

**3. Kullanım**

- Metinsel ifadelerde "" arasına yazılan ifadeler tek satırda yazılmalıdır.

string metin = "Comolokko Aaaaaa

ki ki ki ki mo

leleleele"; // derleyici hatası verir.

string metin = "Comolokko Aaaaaa +

"ki ki ki ki" mo +

"leleleele"; // + ile hata giderilir.

ya da @ ile

string metin = @"Comolokko Aaaaaa

ki ki ki ki mo

leleleele"; //sorunu çözebiliriz.

**String Interpolation İle Verbatim String Birlikteliği (C# 8.0)**

string mailMessage = @"Merhaba ..... .... \n

.... no'lu sipariş talebiniz tarafımızca alınmıştır. \n

Fiyat : ....";

Birlikte kullanmılmak istiyorsa @'dan sonra $ yazılmalıdır.

string isim = "Ali Joe", soyisim = "Yarenchuk", siparisNo = "123123";

int fiyatBilgisi = 150;

string mailMessage = @$"Merhaba {isim} {soyisim}

{siparisNo} no'lu sipariş talebiniz tarafımızca alınmıştır.

Fiyat : {fiyatBilgisi}";

Console.WriteLine(mailMessage);

Çıktı:

Merhaba Ali Joe Yarenchuk

123123 no'lu siparis talebiniz tarafimizca alinmistir.

Fiyat : 150

- Boşlukları da algılar.

string mailMessage =

@$"Merhaba {isim} {soyisim}

{siparisNo} no'lu sipariş talebiniz tarafımızca alınmıştır.

Fiyat : {fiyatBilgisi}";

yazarsak;

Merhaba Ali Joe Yarenchuk

123123 no'lu siparis talebiniz tarafimizca alinmistir.

Fiyat : 150

//daha güzel bir çıktı elde ederiz.

**String Fonksiyonları Nelerdir?**

**Contains**

- Bir metnin içerisinde herhangi bir değerin olup olmadığı kontrol edip bool değer döndüren fonksiyondur.

string metin = "laylay şinanay hop hop smooth criminal hop";

bool sonuc = metin.Contains("hop"); //büyük küçük harf duyarlılığı vardır. "Hop" aratırsak false döner.

CW(sonuc); //true

**StartsWith**

- İlgili metinin verilen değerle başlayıp başlamama durumunu kontrol edip sonuc olarak bool döndürür.

Console.WriteLine(metin.StartsWith("laylay")); //true

EndsWith

- İlgili metinin verilen değerle bitip bitmeme durumunu kontrol edip sonuc olarak bool döndürür.

Console.WriteLine(metin.StartsWith("l hop")); //true

**Equels**

- Elimizdeki metinsel ifade ile herhangi bir ifadenin eşitliğini kontrol edip bool döner.

Console.WriteLine(metin.Equals("Maho")); //false

Console.WriteLine(metin.Equals("laylay şinanay hop hop smooth criminal hop")); //true

**Compare**

- Metinsel ifadeleri karşılaştırır ve sonuçta integer döner. 0 dönerse her iki değer eşit, 1 dönerse soldaki değer alfanümerik olarak sağdakinden büyüktür;, -1 dönerse de küçüktür.

Console.WriteLine(string.Compare(metin, "Z")); //Z > l olduğundan -1 döner.

Console.WriteLine(string.Compare(metin, "a")); //l > a olduğundan 1 döner.

Console.WriteLine(string.Compare(metin, "laylay şinanay hop hop smooth criminal hop")); // 0 döndürür.

Console.WriteLine(string.Compare(metin,metin); //0 döndürür.

Console.WriteLine(string.Compare(metin,9,metin,8,5); //metin'in 9. index'inden başlayıp, ikinci karşılaştırma yapacağımız ifade olan yine metin ifadesinin 8. index'inden başlayarak

//her ikisininde 5 tane karakter al. n ile i kıyasla. n > i olduğundan 1 döner.

**CompareTo**

- Compare ile her şeyi aynı sadece kullanımı farklı.

Console.WriteLine(metin.CompareTo("Z"); //-1 : l < Z

Console.WriteLine(metin.CompareTo("a"); //1 : a > l

Console.WriteLine(metin.CompareTo(metin);//0 : metin = metin

**IndexOf**

string metin = "laylaylom galiba sana göre sevmeler...";

- Verilen değerin string ifade içerisinde olup olmamasını kontrol edip geriye integer olarak indexNo'yu döndürür.

Console.WriteLine("la")); //0

Console.WriteLine("lay")); //0

Console.WriteLine("La")); //-1

Console.WriteLine("sana"));//17

- IndexOf ilk eşleşen değerin index'ini döndürür.

**Insert Metodu**

string metin = "laylaylom galiba sana göre sevmeler...";

- Elimizdeki metinsel ifadeye bir değer dahil etmemizi/eklememizi sağlar.

metin.Insert(int startIndex, string value)

string eklenmisMetin = metin.Insert(17, "merhaba");

- İlgili ekleme operasyonu gerçekleştikten sonra eklenmiş hali string olarak döndürülecektir. Lakin metin orjinal verisi ayrı bir şekilde tutulacaktır. Dönen eklenmiş veriyle

var olan metin arasında fark vardır.

Console.WriteLine(metin));

Console.WriteLine(eklenmisMetin));

Çıktı:

laylaylom galiba sana göre sevmeler...

laylaylom galiba merhabasana göre sevmeler...

**Remove Metodu**

- Metinsel ifadede indexel olaak verilen değer aralığı silen bir fonksiyondur.

metin.Remove(5); //5.indexten sonraki tüm değerleri sil.

metin.Remove(5,10); //5.index'ten başla 10 tane değer sil.

- Insert'te olduğu gibi ilgili fonksiyon yapmış olduğu işlem neticesinde yeni değeri üreterek bizlere string döndürecektir. Elimizdeki orjinal veri değişmeyecektir.

Console.WriteLine(metin.Remove(5));

Console.WriteLine(metin.Remove(5,10));

Console.WriteLine(metin);

Çıktı:

layla

laylaa sana göre sevmeler...

laylaylom galiba sana göre sevmeler...

**Replace Metodu**

- Elimizdeki metinsel ifadede belirtilen değerleri yahut karakterleri, belirtilen diğer değerler ya da karakterle ile değiştirmemizi sağlayan bir fonksiyondur.

Console.WriteLine(metin.Replace('a','b'));

Console.WriteLine(metin.Replace("la","lo"));

Çıktı:

lbylbylom gblibb merhbbbsbnb göre sevmeler...

loyloylom galiba merhabasana göre sevmeler...

- Sonuç olarak string değer üretecek ve geriye döndürecektir.

**Split Metodu**

- Metinsel ifadeyi verilen değeri ayraç olarak kullanıp parçalayan ve sonucu string dizisi olarak döndüren bir fonksiyondur.

string[] dizi = metin.Split(' '); //her boşluğa denk geldiğinde parçala ve parçaları dizinin elemanlarına ata.

string[] dizi = metin.Split(' ','a'); //hem boşluk hem a gördükçe parçala.

**Substring Metodu**

- Metinsel ifadenin belirli bir aralığını elde etmemizi sağlar.

metin.Substring(5); //5.indexten sonuna kadar tüm değerleri getir.

Console.WriteLine(metin.Substring(5,10)); //5. indexten başla sonra 10 karakterti getir.

Çıktı:

ylom galiba merhabasana göre sevmeler111...

ylom galib

**ToLower Metodu**

- Elimizdeki metinin tüm karakterlerini küçük karakter yapar.

string metin = "AAAAAA Kİ Kİ Kİ AAAAAAA";

System.Console.WriteLine(metin);

System.Console.WriteLine(metin.ToLower());

**ToUpper Metodu**

string metin = "aaaaaa ki ki ki aaaaaa";

System.Console.WriteLine(metin);

System.Console.WriteLine(metin.ToUpper());

Çıktı:

aaaaaa ki ki ki aaaaaa

AAAAAA KI KI KI AAAAAA

**Trim Metodu**

- Metinsel ifadelerin varsa, solundaki ve sağındaki karakterlerini temizleyen bir fonksiyondur.

Console.WriteLine("Ahmet Mehmedovski");

Console.WriteLine(" Ahmet Mehmedovski ".Trim());

Çıktı:

Ahmet Mehmedovski

Ahmet Mehmedovski

**TrimEnd Metodu**

- Sadece sağındaki boşlukları temizler.

Console.WriteLine("Ahmet Mehmedovski");

Console.WriteLine(" Ahmet Mehmedovski ".TrimEnd());

Çıktı:

Ahmet Mehmedovski

Ahmet Mehmedovski

**TrimStart Metodu**

- Sadece solundaki boşlukları temizler.

Console.WriteLine(" Ahmet Mehmedovski ".TrimEnd());

Console.WriteLine(" Ahmet Mehmedovski ".TrimStart());

Çıktı:

Ahmet Mehmedovski

Ahmet Mehmedovski //boşluk var.

Örnek:

Adımızın baştan 3, soyadımızın sondan 5. karakteri getirelim.

string adSoyad = "Çağatay Demirtaş";

System.Console.WriteLine(adSoyad[2]);

System.Console.WriteLine(adSoyad[adSoyad.Length - 5]);

Çıktı:

g

i

İkinci Çözüm:

string adSoyad = "Çağatay Demirtaş";

string aralik = adSoyad[2..^4]; //Ranges and Indices

System.Console.WriteLine(aralik[0]);

System.Console.WriteLine(aralik[aralik.Length-1]);

Çıktı:

g

i

**Örnek**:

Girilen Metnin İçerisinde Kaç Adet n Karakterinin Geçtiğini Hesapla

System.Console.WriteLine("Lütfen bir metin giriniz: ");

string metin = Console.ReadLine();

int adet = 0;

for (int i = 0; i < metin.Length; i++)

{

if(metin[i] == 'n')

adet++;

}

System.Console.WriteLine(adet);

Çıktı:

Lütfen bir metin giriniz:

sen ne diyon bana ben seni dinledim sen de beni dinle

10

**Örnek**:

Girilen Metindeki Kelime Sayısını Hesaplayalım.

1.Çözüm

System.Console.WriteLine("Lütfen bir metin giriniz: ");

string metin = Console.ReadLine();

string[] kelimeler = metin.Trim().Split(' ');

System.Console.WriteLine(kelimeler.Length);

Çıktı:

Lütfen bir metin giriniz:

aşkın kanununu yazsam yeniden

4

2.Çözüm

System.Console.WriteLine("Lütfen bir metin giriniz: ");

string metin = Console.ReadLine();

int adet = 1; //bir boşlukta en az 2 kelimemiz olur o yüzden 1'den başla.

while(true)

{

int index = metin.Trim().IndexOf(' '); //baştaki ve sonraki boşlukları da saymasın.

if(index == -1)

break;

adet++;

metin = metin.Substring(index + 1);

}

System.Console.WriteLine(adet);

Çıktı:

we don't need no education

5

Not: String ifadelerde + ile birleştirme maliyet açısından önerilmemektedir.

**DERS-14: ArraySegment ve StringSegment**

Diziler üstünde yapılan çalışmalarda bu tür maliyetleri ortadan kaldırabilmek için veya daha performanslı çalışabilmek uğruna ArraySegment ve StringSegment türlerleri geliştirilmiştir.

- Bu türler yapısal olarak dizi yahut string ifadelerdeki bir bölümü temsil etmemizi sağlayan ve bütünsel açıdan

ilgili veri kümesini parça parça birden fazla referans eşliğinde yönetmemize imkan veren türlerdir.

- İşlemler orjinal array üzerinde referanslarla gerçekleşmekte, ekstradan alan tahsisi veya türetme olmamaktadır.

int[] sayilar = { 10, 20, 30, 40, 50, 60, 70, 80, 90, 100 };

int sayilar2 = sayilar[2..7]; //30 ile 70 aralığını getir. Bir başka diziye ata. Maliyet!

sayilar2 dizisi artık yeni bir dizi. İndex'leri 0'dan başlar.

sayilar2[0] \*= 10;

sayilar2[1] \*= 10;

sayilar2[2] \*= 10;

sayilar2[3] \*= 10;

sayilar2[4] \*= 10;

//Yapılan işlemler orjinal sayilar dizisini etkilemez. sayilar2 dizisinde işlemlerin etkilerini görebiliriz.

**ArraySegment Nedir?**

- Bir dizinin bütününden ziyade belirli bir kısmına yahut parçasına ihtiyaç dahilinde ilgili diziyi kopyalamak yerine

bağımsız bir referans ile erişmemizi ve böylece salt bir şekilde temsil etmemizi sağlayan bir yapıdır.

ArraySegment Türü İle Dizinin Belli Bir Alanını Referans Etmek

int[] sayilar = { 10, 20, 30, 40, 50, 60, 70, 80, 90, 100 };

ArraySegment<int> segment1 = new ArraySegment<int>(sayilar); //ileride daha detaylı

segment1, sayilar dizisinin tüm elemanlarını komple referans etmekte. segment1 üzerinde herhangi bir değişiklik yapıldığında sayilar dizisinde, sayilar üzerinde yapıldığında segment1

üzerinde değişiklikleri görebiliriz. Yeni dizi türetilmez.

ArraySegment<int> segment2 = new ArraySegment<int>(sayilar, 2, 5); //sayilar dizinden 2. index'ten 5. aralığa kadar (30 - 40 - 50 değerleri).

//5. aralık demek soldan 1-2-3-4-5 say demektir.

//sayilar, segment1, segment2'de yapılan değişiklikler birbirlerini etkiler.

segment1[0] \*= 10; //sayilar[0]'da 10\*10 = 100 oldu.

segment2[0] \*= 10; //segment2'nin 0. index'i sayiların 2. index'i yani 30. -> 30 \* 10 = 300

//int[] sayilar => { 100, 20, 300, 40, 50, 60, 70, 80, 90, 100 }

//segment1[] => { 100, 20, 300, 40, 50, 60, 70, 80, 90, 100 }

//segment2[] => dizi yine { 100, 20, 300, 40, 50, 60, 70, 80, 90, 100 } ama referans edilen değer aralığı { 300, 40, 50 }

**ArraySegment Slicing(Dilimleme) Özelliği**

- Bir dizi üzerinde birden fazla parçada çalışılacaksa eğer her bir parçayı ayrı bir ArraySegment olarak tanımlayabiliriz. Bu tanımlamaların dışında diziyi tek bir ArraySegment ile referans edip ilgili parçaları o segment üzerinden talep edebiliriz.

Yani ilgili diziyi tek bir segment üzerinden daha rahat bir şekilde parçalayabiliriz. Bu durum bize yazılımsal açıdan bir efektiflik sağlayacaktır.

int[] sayilar = { 10, 20, 30, 40, 50, 60, 70, 80, 90, 100 };

ArraySegment<int> segment = new ArraySegment<int>(sayilar);

ArraySegment<int> segment1 = segment.Slice(0,3); //integer türünde ArraySegment döndürür.

ArraySegment<int> segment2 = segment.Slice(4,7);

ArraySegment<int> segment3 = segment.Slice(5,10);

- Bu şekilde yapılan çalışmalarda belirli değer aralıklarını elimizdeki ana diziyi tutan/referans eden bir ArraySegment üzerinden dilimleyerek gerçekleyebiliriz. Kod maliyeti açısından da verimlidir.

**StringSegment Türü Nedir?**

- String bir dizi olsa da yapısal olarak kendisine ait fonksiyonlar barındırdığı için, bunlardan kaynaklanan maliyeti absorbe edebilmek amacıyla String'e özel olan StringSegment'i kullanabiliriz.

- Esasında metinsel değerlerdeki birçok analitik operasyonlardan bizleri kurtarmakta ve Substring vs. gibi fonksiyonlar yerine string değerde hedef kesit üzerinde işlem yapmamızı sağlayan bir türdür.

- StringSegment kullanabilmek için kütüphaneye ek bir paket yüklemek gereklidir.

**StringSegment Türü İle Dizinin Belli Bir Alanını Referans Etmek**

string text = "Ölüme gidelim dedin de mazot yok mu dedik.";

StringSegment segment = new StringSegment(text);

StringSegment segment1 = new StringSegment(text, 2, 5);

System.Console.WriteLine(segment); //string segment direkt string'e dönüştürülebildiğinden CW'ile kullanılabilir.

System.Console.WriteLine(segment1);

Çıktı:

Ölüme gidelim dedin de mazot yok mu dedik.

üme g

**StringBuilder Sınıfı Nedir? Ne Amaçla Kullanılır?**

string isim = "Alican";

string soyisim = "Bonomo";

Console.WriteLine(isim + soyisim); //işte bunlar hep maliyet.

- StringBuilder, string birleştirme operasyonlarında + operatörüne nazaran yüksek maliyeti absorbe edebilmek için arka planda StringSegment algoritmasını kullanan ve bu algoritma ile ilgili değerleri olabilecek en az maliyetler birleştirip döndüren bir sınıftır.

StringBuilder kullanabilmek için Using System.Text'i eklemek lazım.

StringBuilder builder = new StringBuilder();

builder.Append(isim);

builder.Append(" ");

builder.Append(soyisim);

Console.WriteLine(builder.ToString());

**DERS-15: Span, ReadOnlySpan, Memory ve ReadOnlyMemory Türleri Nedir? Nasıl Kullanılır?**

**Span Türü nedir?**

- Bellek üzerinde belirli bir alanı temsil ederek işlemler gerçekleştirmemizi sağlayan bir struct'tır.

Bu belirli alanlardan kasıt tabi ki de ardışıl alan kaplayan Array değerleridir.

- Normal şartlarda Array'lerin belleğin HEAP kısmında tutulduğunu biliyoruz. Lakin stackalloc keyword'ü sayesinde STACK'te de Array tanımlayabilmekteyiz.

- Span, stack yahut heap farketmeksizin tanımlanmış olan Array'lerin tümünü yahut bir bölümünü bizler için refere edebilen ve üzerinde işlemler gerçekleştirmemizi sağlayan yapılardır.

- Span, dizi ve string gibi maliyetli veriler üzerinde yapılacak operasyonlarda performans açısından yüksek getiriyle birlikte maliyeti olabildiğince düşürmekte ve ekstradan değer kopyalamaya gerek kalmaksızın tüm faaliyetleri gerçekleştirmemizi sağlamaktadır.

**ReadOnlySpan<T>**

- Span niteliklerinin tümünü sağlamakta adı üzerinde sadece okunabilir kılmaktadır.

- Span ile ArraySegment ve StringSegment farkı: ArraySegment sadece string ve dizilerde temsiliyet yapabilirken,

- Span bellek üzerinde olan herhangi bir yeri temsil edebilir.

- ArraySegment'te referans edilen alana her türlü müdahele edilirken, ReadOnlySpan'da bu verisel operasyonlar engelleneilir ve sadece okunabilir bir davranış sergilenebilir.

- Sadece string yahut array türler ile çalışılacaksa eğer ArraySegment ve StringElement tavsiye edilir.

**Memory Türü nedir?**

- Span ref struct olarak tasarlanmış bir struct'tır. (OOP konusu)

- Dolayısıyla Heap'te allocation(tahsis) edilememe; object, dynamic yahut interface türleri aracılığıyla referans edilmeme; yahut bir class içerisinde field ya da property olarak tanımlanamama gibi kısıtlamaları vardır. (Unut burayı.)

- Memory türü Span'ın bu kısıtlamarından uzak bir versiyonudur.

**ReadOnlyMemory**: Memory türünün sadece okunabilir halidir.

Örnek:

using System;

using System.Text;

using Microsoft.Extensions.Primitives;

namespace OrnekProje

{

class Program

{

static void Main(string[] args)

{

int[] sayilar = { 10,20,30,40,50,60,70,80,90,100 };

//tüm diziyi referansladık.

Span<int> span = new Span<int>(sayilar);

//tüm diziyi referansladık.

Span<int> span2 = sayilar;

Span<int> span3 = new Span<int>(sayilar, 3, 5); //3. index'ten başla 5 tane değer al.

//span1,span2 ve span3 aynı diziyi refere eder. Üzerlerinde yapılan işlem orjinal array'i etkiler.

Span<int> span4 = sayilar.AsSpan(); //diziden span türünde bir nesne talep edebiliriz.

Span<int> span5 = sayilar.AsSpan(3,5); //ya da belirli bir aralığı belirtebiliriz.

string text = "Sen kalbimde batan güneş, ben yollarda çilekeş";

//metinsel ifadeler üzerindeki referanslar readonly gelecektir!

ReadOnlySpan<char> readOnlyspan = text.AsSpan();

ReadOnlySpan<char> span6 = text;

}

}}

**DERS-16: Regular Expressions**

- Metinsel yapılanmalarda belirli koşulları sağlayabilen ifadelerdir.

- Bir metinsel ifade içerisinde @ karakteri geçen bütün aralıkları elde etmek istediğimizi varsayalım.

bdasbdsabd@as14234jasd@adsksdfj123''4asdfff

- Dikkat ederseniz bu değerlerdeki karakterlerin uzunluğu ve ne olduğu önemli değil. Yeter ki @ karakteri olsun.

- Bunu normal kod yazımı(loop'lar vs.) yollarıyla yapmak ileri seviye, karışık programlarda zor olacaktır.

- Milyonlarca e-mail adresi olduğunu düşünelim. Biliyoruz ki tek bir e-mail formatı var.

Her e-mail mutlaka @ ve ardından .(nokta) karakterlerini barındırır. Eğer birden fazla nokta varsa, noktalardan biri mutlaka @ karakterinden sonra olmalıdır.

Haliyle bir karakter dizisinin e mail adresi olup olmadığını test etmek oldukça zor olacaktır.

Bu yüzden bu düzeni kavrayabilme işini Regular Expressions'lara bırakıyoruz.

**Regex**

- C#'ta bu tür düzenli ifadeleri temsil edebilmek için Regular Expressions operatörleri geliştirilmiştir.

- Bu operatörler eşliğinde elde edilen verinin tasarlanan metinsel düzene uyup uymadığı değerlendirilebilmektedir.

- Regular ifadeler System.text.RegularExpressions namespace'i altındaki Regex sınıfı tarafından temsil edilmektedir.

- Regular Expression'lar ufak tefek farklılıklar olsa dahi tüm programlama dillerinde mevcuttur.

**Regular Expressions Operatörleri ^ Operatörü, Regex Sınıfının Kullanımı**

^ Operatörü

- Satır başının istenilen ifadeyle başlamasını sağlar.

- Using keyword'ü ile using System.Text.RegularExpressions; eklemek lazım.

^9 : Yazı 9 ile başlamalıdır. 9aafsdf, 912312313, 9'''114!++ hepsi uygundur. 123asdf, 'dasfsdf uygun değildir.

string text = "9123123234adsdfsdfsdf";

Regex regex = new Regex("^9");

//regex'in içerisindeki pattern neyse o pattern'a sahip regex üzerinden Match'i çağırdığımızda verilen text değeri ne ise kontrol edilecek pattern'a uygunluğu belirtilecektir. Sonuç Match sınıfıyla döner.

Match match = regex.Match(text);

System.Console.WriteLine(match.Success); //True yazacaktır. 9 ile başlamasaydı False verirdi.

**\ Operatörü**

\ : Belirtilen karakter gruplarını içermesini istiyorsak.

\D : Metinsel değerlerin ilgili yerinde rakam olmayan tek bir karakterin bulunması gerektiğini belirtir.

\d : Metinsel değerlerin ilgili yerinde 0 - 9 arasında tek bir sayı olacağı ifade edilir. (tek-sadece)

\W : Metinsel değerin ilgili yerinde alfanümerik olmayan karakterin olması gerektiği belirtilir. Alfanümerik karakterler: a-z A-Z 0-9

\w : Metinsel değerin ilgili yerinde alfanümerik olan karakterin olacağı ifade edilir.

\S : Metinsel değerin ilgili yerinde boşluk karakterleri (tab/space) dışında herhangi bir karakterin olamayacağı belirtilir.

\s : Metinsel değerin ilgili yerinde sadece boşluk karakterinin olacağı ifade edilir.

Örnek:

//9 ile başlayani ikinci krakteri herhangi bir sayı olan ve üçüncü karakteri de boşluk olmayan bir düzenli ifade oluşturalım.

//^9\d\S

string text = "921231423442sfdsdfsdfsdg";

Regex regex = new Regex(@"^9\d\S");

//verbatim ile özel ifadeleri kullanabiliriz. Birinden biri olmazsa bile False döndürecektir.

Match match = regex.Match(text);

System.Console.WriteLine(match.Success);

Çıktı:

True

**+ Operatörü**

- Belirtilen gruptaki karakterlerden bir ya da daha fazlasının olmasını istiyorsak kullanabiliriz.

//9 ile başlayan, arada herhangi bir sayısal değerleri olan ve son karakteri de boşluk olmayan bir düzenli ifade oluşturalım.

//^9\d+\S -> 9 ile başlaması, ikinci ve ikinciden daha fazla karakterin sayısal olması ve bundan sonraki karakterin boşluk olmaması

string text = "921231423442sfdsdfsdfsdg";

//"9874a23424d -> True

//"1234ffdsfdd -> False

//"9a1234fszxx -> False

//"98 58ffasds -> False

//"9854 asdasd -> True

Regex regex = new Regex(@"^9\d+\S"); //verbatim ile özel ifadeleri kullanabiliriz.

Match match = regex.Match(text);

System.Console.WriteLine(match.Success); //true

**|(veya) Operatörü**

Birden fazla karakter grubunun bir ya da birkaçının ilgili yerde olabileceğini belirtmek istiyorsak mantıksal veya operatörü kullanılır.

Örnek:

Baş harfi a ya da b ya da c olan metinsel ifade.

a| b | c

string text = "Ahmet"; //False

//string text = "ahmet"; True

Regex regex = new Regex(@"a| b | c");

Match match = regex.Match(text);

Console.WriteLine(match.Success);

**{n} Operatörü**

- Sabit sayıda karakterin olması isteniyorsa {adet} şekline belirtilmeli.

//532-3454564

//\d{3}-\d{7} ->üç tane rakam olsun

string text = "532-3454564"; //True

Regex regex = new Regex(@"\d{3}-\d{7}");

Match match = regex.Match(text);

Console.WriteLine(match.Success);

**? Operatörü**

- Bu karakterin önüne gelen karakter metinde en fazla bir defa geçmeli ya da hiç geçmemelidir.

// \d{3}B?A --> ilk üç ifade sayısal olacak, sonraki ifadede enfazla 1 tane B geçmeli ya da geçmese de olur, sonrakinde de A olacak.

string text = "234BA"; //True

//"543BA" -> true

//"543A" -> true

//"123BBA" -> False , B iki kere geçiyor.

Regex regex = new Regex(@"\d{3}B?A");

Match match = regex.Match(text);

Console.WriteLine(match.Success);

**. Operatörü**

\n hariç herhangi bir karakterin kullanılabileceğini belirtir.

Örnek:

//d{3}.A ->ilk üçü sayı, sonrki herhangi bir karakter, ardından A değeri gelecek

string text = "123'A'"; //True

//string text = "123\nA'"; //False

Regex regex = new Regex(@"d{3}.A");

Match match = regex.Match(text);

Console.WriteLine(match.Success);

\b - \B Operatörleri

\B : Bu ifade ile kelimenin başında ya da sonunda olmaması gereken akrakterler bildirilir.

\b : Bu ifade ilgili kelimenin belirtilen karakter dizisi ile sonlanmasını sağlar.

// \d{3}dır\B -> üç tane sayı olsun ama sonunda ya da başında dır eki olmasın. 123dır olmaz, dır123 olmaz. 123dır2 olur.

string text = "123dır";

Regex regex = new Regex(@"\d{3}dır\B");

Match match = regex.Match(text);

Console.WriteLine(match.Success);

**[n] Operatörleri**

- Metinsel ifadede belirli bir noktada belirli karakter aralığındaki değerlerin olmasını istiyorsak eğer [n] kullanabiliriz.

Ayrıca özel karakterin yerinede yazılmasını da ifade eder.

// \d{3}[A-E] -> ilk 3 sayı, sonrakinde A-E arası bir karakter bekle

string text = "123A"; //true

//"123C" -> true

//"123P" -> false

Regex regex = new Regex(@"\d{3}[A-E]");

Match match = regex.Match(text);

Console.WriteLine(match.Success);

//özel karakterler

//(530) 751 33 83 formatında tel no almak istiyorsak. -(- ve -)-'ı [] içerisine almamız lazım.

//[(]d{3}[)]\s\d{3}\s\d{2}\s\d{2}

string text = "(530) 751 33 83"; //true

Regex regex = new Regex(@"[(]d{3}[)]\s\d{3}\s\d{2}\s\d{2}");

Match match = regex.Match(text);

Console.WriteLine(match.Success);

**Regular Expressions Match Sınıfı Özellikleri**

string text = "(530) 751 33 83"; //true

Regex regex = new Regex(@"[(]\d{3}[)]\s\d{3}\s\d{2}\s\d{2}");

Match match = regex.Match(text);

Console.WriteLine($"Success : {match.Success}");

Console.WriteLine($"Value : {match.Value}");

//doğrulanan verinin ne olduğunu getirir. "(530) 751 33 83asdsadsadff2231" olsaydı bile sadece (530) 751 33 83 getirirdi.

Console.WriteLine($"Index : {match.Index}"); //doğrulamanın hangi index'te başladığını bildirir.

Console.WriteLine($"Length : {match.Length}"); //doğrulamanın yapıldığı ifadenin kaç karakterli olduğunu bildirir.

Çıktı:

Success : True

Value : (530) 751 33 83

Index : 0

Length : 15

**DERS-17: Koleksiyonlar ve Foreach**

**ArrayList Koleksiyonu**

- Koleksiyonlara geçişlerde ilk ara geçiş örneği gibi düşünülebilecek bir koleksiyondur.

ArrayList kullanabilmek için using System.Collections; kodun en üstüne eklemek gerekli.

**Tanımlama**

int[] yaslar = new int[17];

ArrayList \_yaslar = new ArrayList();

//Diziler tanımlanırken kaç eleman tutacakları bildirilmek zorundadır.

//Buradaki sınırlılık ArrayList'te kalkıyor.

**Değer Atama**

yaslar[5] = 123; //dizide değer atama

\_yaslar.Add(123); //koleksiyonlarda değer atama (pek çoğunda Add kullanılır.)

**ArrayList Tanımlanmış Koleksiyondan Değer Okuma**

for (int i = 0; i < 17; i++)

{

yaslar[i] = i + 10;

\_yaslar.Add( i+10);

}

//Diziler

System.Console.WriteLine(yaslar[5]);

//Koleksiyonlar

System.Console.WriteLine(\_yaslar[5]);

Çıktı:

15

15

**ArrayList Boxing - Unboxing Durumlarından Dolayı Sınırlılıklar**

- ArrayList verilen datayı otomatik olarak boxing işlemine tutarak, object olarak değer alır. Bu verileri kendi türlerinde elde etmek için Unboxing yapmamız gerekir. Ahanda maliyet! İleride Generic Fonksiyonlar ile bu maliyetleri aşabileceğiz.

Örnek: //Aritmetik ortalama

using System;

using System.Collections;

using System.Text;

using System.Text.RegularExpressions;

using Microsoft.Extensions.Primitives;

namespace OrnekProje

{

class Program

{

static void Main(string[] args)

{

ArrayList \_yaslar = new ArrayList();

for (int i = 0; i < 17; i++)

{

\_yaslar.Add(i+10);

}

int toplam = 0;

int arOrt = 1;

for (int i = 0; i < \_yaslar.Count; i++) //Koleksiyonlarda Count ile eleman sayısını öğreniriz.

{

toplam += (int)\_yaslar[i] ;

}

arOrt = toplam / \_yaslar.Count;

System.Console.WriteLine($"0. Index: {\_yaslar[0]} \n16.Index: {\_yaslar[16]}");

System.Console.WriteLine($"Toplam: {toplam}" );

System.Console.WriteLine($"Aritmetik Ortalama: {arOrt}");}}}

**ArrayList Collection Initializers(Koleksiyon İlklendirici)**

ArrayList arrayList = new ArrayList() {

//object değil mi yapıştır her türden.

"Ahmet",

123,

'a',

true

}; //bu işlemin adı koleksiyon ilklendiricidir.

//Bu değerlerin hangi türden olduğunu kontrol edip ona göre işlemleri yapmak lazım. if(\_yaslar[i] is int)... tarzı kontroller yapılmalı.

**İterasyon Nedir?**

- Mantıksal açısından her tahminin altında bir iterasyon mantığı yatar.

1

3

5 sonra ne gelebilir?

7 -> gelir diyorsan iterasyon mantığı kullanmışsındır.

f(5) = 10

f(10)= 20

f(15)= 30 ise f(20) = 40'dır diyebiliriz.

- Dil biliminde de iterasyonel kelimeler vardır. baba, mermer gibi.

- Eğer ki veri kümeleri içerisinde iterasyonel operasyonlar sağlıyorsak, bu kümenin içerisindeki bütün değerleri tek tek elde etmek zorundayız. Yüzdelik, kısımsal işlemlere pek uygun değildir. İterasyon her veriyi tek tek alacaktır, aldıktan sonra işlem anında o veriyi tanıyıp, veriyle işlem yapılıp yapılmayacağına kişinin karar vermesi lazım.

**Foreach İterasyonu**

- Elimizde bir veri kümesi yani array ya da koleksiyon olması şart.

foreach(\_\_\_\_\_\_değişken\_\_\_\_\_\_\_ in \_\_\_\_\_\_\_collection or array\_\_\_\_\_\_\_\_)

{

}

Örnek:

ArrayList sayilar = new ArrayList {123,123,325,534,5,345,345};

foreach (object item in sayilar)

{

System.Console.WriteLine(item);

}

Çıktı:

123

123

325

534

5

345

345

Örnek:

ArrayList sayilar = new ArrayList {123,123,325,534,5,345,345};

foreach (object item in sayilar)

{

sayilar.Add(123); //koleksiyona eleman/veri ekledik ve yapısını bozduk. foreach ile çalışırken veri kümesinde bir //değişiklik olursa hata verir. Çalışmaz.

//o yüzden genelde kaynakta değişiklik yapmamak lazım.

System.Console.WriteLine(item);

}

Örnek:

For ile verilen kodu foreach'e dönüştürelim.

ArrayList grades = new ArrayList() {5,10,26,34,26,26,2,23,26,200};

for (int i = 0; i < grades.Count; i++)

{

System.Console.WriteLine((int) grades[i] \* 2);

}

Foreach ile;

foreach (int grade in grades)

{

System.Console.WriteLine(grade);

}

//2 ile çarpma işlemini veri kümesinde değişiklik yapmamak için kullanmadık. Fakat foreach'i veri okumak için kullanabiliriz.

**DERS-18: Hazır Sınıflar ve Metotlar**

**Math Sınıfı - Abs Fonksiyonu**

- Using.System'ı kullanır. Ekstra eklemeye gerek yok.

- Abs, mutlak değer işlemi yapar. Absolute value'dan gelir.

int i = Math.Abs(-5);

**Math Sınıfı - Ceiling Fonksiyonu**

- Herhangi bir ondalıklı değeri yukarıya yuvarlar.

CW(Math.Ceiling(3.14)); // 4 olur.

**Math Sınıfı - Floor Fonksiyonu**

- Herhangi bir ondalıklı değeri aşağı yuvarlar.

CW(Math.Floor(3.14)); // 3 olur.

**Math Sınıfı - Round Fonksiyonu**

- Ondalıklı değeri kendisine en yakın olan tamsayıya yuvarlar.

CW(Math.Round(3.4)); // Aşağı yuvarlar. 3 olur.

CW(Math.Round(3.5)); // Yukarı yuvarlar. 4 olur. Tam ortadaki değerler yukarı yuvarlanır.

CW(Math.Round(3.6)); // Yukarı yuvarlar. 4 olur.

**Math Sınıfı - Pow Fonksiyonu**

- Sayının üssünü almaya yarar.

CW(Math.Pow(7,9)); //7 üssü 9 ve sonuç olarak double döner.

**Math Sınıfı - Sqrt Fonksiyonu**

- Karakök almaya yarar.

CW(Math.Sqrt(4)); //2

CW(Math.Sqrt(16)); //4

CW(Math.Sqrt(55)); //7.416...

**Math Sınıfı - Truncate Fonksiyonu**

- Ondalıklı değerlerin sadece tamsayı kısmını elde etmekte kullanılır.

CW(Math.Truncate(3.14)); //3'ü döndürür.

**DateTime - Now Özelliği**

CW(DateTime.Now)); //o anki tarih ve saati verir. 27.09.2022 15:06:31

**DateTime - Today Özelliği**

- Sadece tarih bilgisini verir.

CW(DateTime.Today)); //27.09.2022 00:00:00

**DateTime - Compare Fonksiyonu**

- Verilmiş iki tarihi kıyaslar ve integer değer döndürür.

DateTime tarih1 = new DateTime(2021, 01, 01);

DateTime tarih2 = new DateTime(2022, 01, 01);

int result = DateTime.Compare(tarih1,tarih2);

if(result < 0) //birinci tarih ikincisinden geride.

{

System.Console.WriteLine($"{tarih1} küçüktür {tarih2}");

}

else if(result == 0)

{

System.Console.WriteLine($"{tarih1} eşittir {tarih2}");

}

else

System.Console.WriteLine($"{tarih1} büyüktür. {tarih2}");

Çıktı:

1.01.2021 00:00:00 küçüktür 1.01.2022 00:00:00

**DateTime - Tarihsel Zamana Saat, Gün, Ay, Yıl Ekleyerek Sonucu Hesaplamak**

AddYears

System.Console.WriteLine(DateTime.Now.AddYears(999)); //şimdiki zamana 999 yıl ekledik. 27.09.3021 15:20:40

AddSeconds

System.Console.WriteLine(DateTime.Now.AddSeconds(999)); //999 saniye ekledik. 27.09.2022 15:38:41

AddMinutes, AddMonths, AddDays.....

**TimeSpan Türü İle İki Tarih Farkının Karşılanması**

- İki tarih arasındaki farkı timespan türüyle karşılayabiliriz.

DateTime t1 = DateTime.Now;

DateTime t2 = new DateTime(2000,1,1);

TimeSpan span = t1-t2;

System.Console.WriteLine(span.Days);

System.Console.WriteLine(span.Minutes);

Çıktı:

8305

28

**Random Sınıfı - Next Fonksiyonu**

- Rastgele sayısal değer oluşturur.

Random random = new Random();

//belirtilen maksimum değer aralığa dahil değildir.

System.Console.WriteLine(random.Next()); //0 ile int max değeri aralığında rastgele değer üretir.

System.Console.WriteLine(random.Next(100)); //sıfır ile 100 arası

System.Console.WriteLine(random.Next(50,100)); //50 ile 100 arası

- Sadece pozitif değer üretir. Negatif üretmek istiyorsak -1 ile çarpmamız lazım.

**Random Sınıfı - NextDouble Fonksiyonu**

- 0 ile 1 arası rastgele sayı üretir. 0,5921208223291304 gibi.

System.Console.WriteLine(random.NextDouble());

**DERS-19: Metot nedir?**

- Yöntem, yordam, function...

- Yazılımda fiili olarak gerçekleştirdiğimiz tüm operasyonlar bir metot'dur.

- Metot, prosedürel programlamanın temel elemanıdır. Bir iş/fiiliyat/operasyon yapan en küçük program parçacıklarıdır.

- Metot'lar tekrara düşmemizi önler.

int[] sayilar = new int[10];

for (int i = 0; i < sayilar.Length; i++)

{

sayilar[i] = new Random().Next();

}

foreach (var sayi in sayilar)

{

System.Console.WriteLine(sayi);

}

//Aynı kodu 3 kere çalıştırmak istiyorsak; kodu üç kere alt alta yazabiliriz. 3 kere dönen bir for loop'una alabiliriz...

Ya da tekrar eden kodu metot'a alabiliriz. Ve ihtiyaç anında istediğimiz kadar çağırabiliriz.

Böylece tekrarı önleyip ideal kodlamaya ulaşabiliriz.

Metot Anatomisi Nasıldır? Metot İmzası nedir?

- Metot'lar sınıf elemanıdır. Sınıf/class içerisinde çalıştırmamız gerekir.

namespace OrnekProje

{

class Program

{

//Burada bir metot oluşturulabilir.

static void Main(string[] args) //main'de bir metot'dur.

{

//Şimdilik metot içerisinde metot oluşturamıyoruz diye kabul edelim. (ileride local functions konusunda)

}

//Burada da oluşturulabilir.

}

}

[erişim belirleyicisi/access modifier] [geri dönüş değeri] [Metotun adı] () //bu kısım metodun imzasıdır.

{

//burası ise gövdesi olarak geçer.

}

- Erişim belirleyicisi/access modifier'lar ileride göreceğimiz yapılardır. Oluşturulmuş olan bir yapının(illa ki metot olmasına gerek yok) dışarıdan erişilip

erişilmeyeceğini belirtmemizi sağlayan komutlardır. Daha da detaylandırılacak.

public : Erişilsin

private : Gizli/erişilmesin (default)

- Metot'lar yaptıkları operasyonlar/algoritmalar/işlemler neticesinde geriye değerler dönebilir. Bu değer kodun içerisinde yakalanabilir ve programlamatik olarak

işleme tabii tutulabilir. Geri dönüş değerini ekran çıktısı ile karıştırmamak gerekir.

- Metot'larda diğer yapılarda olduğu temel isimlendirme kurallarına uymak kaydıyla, isimlendirilebilir. Rakamla başlayamama, özel karakter vs..

- () ise Metot dediğimiz yapılanmalar dış dünyadan parametreler alabilmekte ve bu parametrelerdeki değerler üzerinde işlemler gerçekleştirebilmektedir.

İşte bu parametreleri parantez içerisinden alabilmekteyiz. Bir metot illa ki parametre almak zorunda değildir. Ama alacaksa türü/cinsi belirtilmelidir.

- {scope} ise Metot'un gerçekleştireceği operasyonu/algoritmayı oluşturduğumuz faaliyet alanıdır.

static void Main(string[] args) //main'de bir metot'dur.

//İşaretleme olmadığı için private. Geri dönüş değeri void, yani olmayan. Main adlı (string[] args) parametresi alan bir metot'dur.

**İşlevine Göre Metot Türleri Nelerdir?**

- Yapılacak işleme göre 4 farklı varyasyonda fonksiyon oluşturabiliriz.

1) Geriye Değer Döndürmeyen, Parametre Almayan

2) Geriye Değer Döndürmeyen, Parametre Alan

3) Geriye Değer Dönen, Parametre Almayan

4) Geriye Değer Dönen, Parametre Alan

**Geriye Değer Döndürmeyen, Parametre Almayan Metotlar**

namespace OrnekProje

{

class Program

{

static void Main(string[] args) //main'de bir metot'dur. İşaretleme olmadığı için private. Geri dönüş değeri void, yani olmayan. Main adlı (string[] args) parametresi alan bir metot'dur.

{

}

//[erişim belirleyicisi/access modifier] [geri dönüş değeri] [Metotun adı] (......)

//{

//}

**//Geriye Değer Döndürmeyen, Parametre Almayan Metotlar**

//bir metot geriye değer döndürmüyorsa bunun void ile döndürülmesi zorunludur.

private void Metot1() //private ya da public oluşuyla bir alakası yok. İstersen public yap.

{

System.Console.WriteLine("Geriye değer döndürmeyen, parametre almayan metot oluşturuldur.");

}

}

}

**Geriye Değer Döndürmeyen, Parametre Alan Metotlar**

namespace OrnekProje

{

class Program

{

static void Main(string[] args)

{

}

//[erişim belirleyicisi/access modifier] [geri dönüş değeri] [Metotun adı] (......)

//{

//}

**//Geriye Değer Döndürmeyen, Parametre Alan Metotlar**

public void Metot2(int a) //parametreler bildiğimiz değişkenlerdir.

{

}

public void Metot3(int a, bool b) //oluşturduğumuz parametre değerlerine metodun scope'undan erişebiliriz.

{

// a değeri metod3'ün parametresinin a'sıdır.

}

public void Metot4(int a, bool b, char c) //sınırsız sayıda parametre değeri girilebilir.

{

}

//Bir "değişken" metodun parametre kısmında tanımlanıyorsa parametre, bu "değişken" class'ın içinde ya da metodun içinde tanımlanıyorsa değişken denilecektir.

//hatta OOP'de class içerisinde tanımlanan değişkenlere field denilecektir.

}

}

**Geriye Değer Döndüren, Parametre Almayan Metotlar**

namespace OrnekProje

{

class Program

{

static void Main(string[] args)

{

}

//[erişim belirleyicisi/access modifier] [geri dönüş değeri] [Metotun adı] (......)

//{

//}

private char Metot5() //eğer ki bir metot geriye herhangi bir türde değer döndüreceğini ifade ediyorsa kesinlikle o türde bir değer döndürmelidir. Aksi takdirde hata verir.

{

//....bişiler....bişiler

return 'a'; //geriye değer döndüren fonksiyonlarda, bildirilen türde bir değer döndürebilmek için return keyword'ü gerekmektedir.

}

//return keyword'ü nerede tetiklenip işlenirse orada ilgili fonksiyondan çıkılır.

private int Metot5()//hata

{

if(DateTime.Now > 10) //hata verir. Çünkü false olduğu durumlarda da geriye bir değer döndürmesi lazım.

return 5;

//return 123; 'ü eklersek hata ortadan kalkar.

}

char metod7() //de yazılabilir. Default olarak private olacaktır.

{

return 'b';

}

}

**Geriye Değer Döndüren Parametre Alan Metotlar**

namespace OrnekProje

{

class Program

{

static void Main(string[] args)

{

}

//[erişim belirleyicisi/access modifier] [geri dönüş değeri] [Metotun adı] (......)

//{

//}

public bool Metot8(int x)

{

return false;

}

public int Metot9(int x)

{

return DateTime.Now.Year > 2000 ? 1 : 0 ; //şimdiki yıl 2000'den büyükse 1 değilse 0 dön.

}

}

}

**Metodun Geriye Değer Döndürmesi Ne Demektir?**

- Bir metodun geriye değer döndürmesi demeki içeride yapılan işlem neticesinde üretilen sonuçların ekrana yazılması demek değildir.

- Metodun geriye döndürdüğü değer, programatik olarak yakalanıp algoritmaları akışında farklı yönlendirmelere sebebiyet verebilen değerdir. Yani bu değer algoritmanın akışında kullanılabilir değerdir.

Örnek: //kurcalayalım.

namespace OrnekProje

{

class Program

{

static void Main(string[] args)

{

bool sonuc = PersonelEkle("Rıfkı", "Conocan", 25);

if (sonuc)

{

.....

}

else

{

.....

}

}

static public bool PersonelEkle(string adi, string soyadi, int yas)

{

if (yas >= 20)

{

//..veritabanına ekle mesela

return true;

}

else

{

System.Console.WriteLine("Personel 20'den küçük olamaz.");

return false;

}

}

static public int Topla(int sayi1, int sayi2) //main'de erişebilmek için static yaptık.

{

int sonuc = sayi1 + sayi2;

System.Console.WriteLine(sonuc);//ekrana verilen çıktı geriye dönülen değer demek değildir.

return sonuc;

}

}

}

**Metotlarda Optional Parameters(İsteğe Bağlı Parametreler)**

- Parametreli bir metot kullanılırken/çağrılırken/tetiklenirken parametrelerine, türlerine uygun değerler gönderilmek zorunludur.

- Eğer ki bir metodun parametrelerine zorunlu bir şekilde değer göndermek istemiyorsak, parametreye değeri isteğimize göre/opsiyonal olarak göndermek istiyorsak, o parametrenin bu durumu karşılayabilecek özellikte olması gerekmektedir. İşte bu özelliğe opsiyonel parametreler denmektedir.

- Bir parametrenin opsiyonel olması demek o parametrenin varsayılan/default değeri olması demektir.

namespace OrnekProje

{

class Program

{

static void Main(string[] args)

{

X(15,20); //b'yi gönderedebiliriz, göndermeyedebiliriz.

X(5); //de yapılabilir artık. b opsiyonel. 0 varsayılan değeri alınacak.

}

static public void X(int a, int b = 0) //metot parametrelerine = assign operatörü ile bir değer atanırsa eğer o parametreye varsayılan değeri atanmış olur. Haliyle opsiyonel parametre haline getirilmiş olunur.

{

//a opsiyonel değil, b opsiyonel oldu.

}

}

}

//static public void X(int a = 5, int b = 0) //hem a hem b'de opsiyonel yapılabilir.

//o zaman da X(); olarak kullanabiliriz.

//Birden fazla parametre durumunda opsiyonel olanlar sağ tarafta TANIMLANMALIDIR!

//static public void X(int a = 5, int b) -> olmaz!

//static public void X(int a, int b = 0, int c = 0) ->olur!

//static public void X(int a, int x, int y, int b = 0, int c = 0) -> olur!

//static public void X(int a, int x, int b = 0, int y, int c = 0) -> olmaz!

**Tanımlanmış Metodun Kullanımı - Tanımlandığı Sınıf İçerisindeki Metotlar Tarafından Kullanımı**

- Bir metot tanımlandığı sınıf içerisindeki farklı bir metot içerisinden çağrılacaksa eğer tek ypaılması gereken sadece isminin çağrılmasıdır.

class Program

{

static void Main(string[] args)

{

X();

}

static void X()

{

}

}

class Ornek

{

public void A()

{

B();

}

private void B()

{

C(5);

}

private int C(int a = 0)

{

return a;

}

}

**Başka Sınıfta Tanımlanmış Metotların Erişimi-Referans ve Nesne İlişkisi**

- 3 bir int'dir. Ahmet bir string'dir.

Adi, Soyadi ,yas, medeniHal, gozRengi...vs... hepsi bir Nesne'yi oluşturur. Karmaşık değerlerin bütünü, Nesne'leri oluşturmamızı sağlayan yapının adı da class'tır.

- Class Model ise, Nesne Class'ın verisidir.

class Ogrenci

------------------

string adi; |

string soyadi;|

int yasi; |

int sinif; |

------------------

- Bir öğrenciyi modelledik. Bir çğrenci tek başına adi ya da yasi değil ama bunların bir araya gelmiş halidir.

Bu modelden bir adet nesne oluşturalım.

adi = "Ahmet";

soyadi = "Yılmaz";

yasi = 20;

sinif = 3;

Class'tan bir değer oluşturduk, ve kompleks değere Nesne-Object dedik. Bu nesne bizim kompleks değerimizin verisidir.

İstediğimiz kadar oluşturabiliriz. Mehmet için de oluşturalım.

adi = "Mehmet";

soyadi = "Yıldız";

yasi = 21;

sinif = 2;

.

.

.

public void X() //bu metotlarla bu class'ın içini istediğimiz gibi manipüle edebiliriz.

{

}

Bu nesnelerin içerisinde değerler olduğu gibi metotlar da olabilir.

Farklı class'larda oluşturulmuş metotlara erişebilmemiz için o sınıflardan nesne oluşturmamız gerekiyor. Bunu da new keyword'ü ile yaparız.

Type isim = new Type() gibi.

Random r = new Random(); 'da kullanmıştık. Random sınıfı üzerinde işlem yapabilmek, onun metotlarına erişebilmek için bir nesne oluşturuyoruz. new diyerek Random'dan bir nesne oluştur. Bu nesneye rahat bir şekilde erişebilmek için bu nesneyi aynı türde bir değişkenle işaretle.

Random türünde r diye bir değişken oluşturduk ardından new ile Random() yazarak Random türünde bir nesne oluşturduk ve r değişkeni üzerinden bu nesne içerisindeki metotlara erişilebilir vaziyette bir nesne elde etmiş olduk.

- Referans oluşturulan nesnenin işaretlenmesini sağlayıp o nesneye de benim erişmemi sağlayan bir değişkendir.

- İleride OOP dersinde hepsi detaylandırılacak.

Özet:

Sınıf - Class : Yeryüzündeki herhangi bir olguyu modellememizi sağlayan yapılanmadır. İçerisinde ilgili olguya dair verileri tutacak alanları(field) barındıran ve bu alanlar üzerinde işlem yapmamızı sağlayacak olan metotları barındıran bir yapı.

Nesne : Class'tan üretilen değer/veri.

Referans : Class'tan üretilen değeri kullanmamızı sağlayan yapı.

class A

public void X()

{

}

-> Bu class'tan X metoduna erişmek istiyorsak,

A a = new A();

referans - nesne (object)

//bu class'tanbir değişken oluştur. Ve bu değişkenin karşılığı olarakta ilgili class'tan new operatörü ile bir nesne oluştur.

//= de artık assign değil referans etme operatörüdür.

Erişmek istiyorsak A a = new A(); yapısının olduğu her yerde a.X ile erişebiliriz. (public ise.)

**Başka Sınıfta Tanımlanmış Metotların Erişimi**

namespace OrnekProje

{

class Program

{

static void Main(string[] args)

{

//Başka Sınıfta Tanımlanmış Bir Metodun kullanımı

Matematik matematik = new Matematik();

//Matematik türünde matematik isminde bir referans oluşturduk. = Matematik türünde bir nesne oluşturduk.

//oluşturulan bu nesneyi Matematik türündeki matematik referansı ile işaretledik.

System.Console.WriteLine(matematik.Carp(3,5)); //public olmsaydı erişemezdik.

}

}

class Matematik

{

public int Topla(int sayi1, int sayi2)

{

return sayi1 + sayi2;

}

public int Cikar(int sayi1, int sayi2)

{

return sayi1 - sayi2;

}

public int Carp(int sayi1, int sayi2)

{

return sayi1 \* sayi2;

}

public int Bol(int sayi1, int sayi2)

{

return sayi1 / sayi2;

}

}

}

**Metotlarda Non Trailing Named Arguments Özelliği**

namespace OrnekProje

{

class Program

{

static void Main(string[] args)

{

X(3, 5, "abc"); //normalde sıraya uygun olmak zorunda. Ama sıraya uymamak istiyorsak;

//Non-Trailing Named Arguments özelliğini kullanabiliriz.

X(c:"abc", a:5, b:5); //gibi istediğimiz sırada değer gönderebiliriz.

//karışıklığa sebep olmaması için bu özellik kullanılıyorsa tüm parametreler için kullanılmalı.

//hangi parametreye hangi değeri gönderdiğimizi görmek için faydalıdır.

//göreceli bir şekilde çok parametreli fonksiyonlarda hedef parametrelere değer göndermemizi sağlar.

}

static void X(int a, int b, string c)

{

}

}

}

**Metotlarda In Parametreleri (C# - In Keyword'ü)**

namespace OrnekProje

{

class Program

{

static void Main(string[] args)

{

//1) Parametrenin değerini metodun içerisinde herhangi bir yerde çağırıp kullanabiliriz.

//2) metot içerisinde üretilen herhangi bir değeri tutacak değişken oluşturmaktansa parametre üzrinde bu değeri //tutabiliriz.

//yani parametrenin değerini değiştirebiliriz. Çünkü parametreler özünde bir değişkendir.

//In komutu sayesinde parametreye verilen değeri sabi tutabilmekteyiz.

//In komutu metodun parametresini readonly hale getirir.

}

static void X(in int a, int b, in char c)

{

a = 123; //derleyici hatası! -in- dediğimiz için değişmez.

b = 12; //sıkıntı yok. çalışır.

c = 'c'; //sıkıntı büyük! in char! olmaz.

} } }

**Local Functions(Metot İçerisinde Tanımlanabilir Yerel Metotlar)**

namespace OrnekProje

{

class Program

{

static void Main(string[] args)

{

X();

//Local Functions

//Bir metot içerisinde tanımlanmış olan metotlardır.

//C#'da metotlar sadece class içerisinde tanımlanırlar diye belirtmiştik. Halbuki OOP'de göreceğimiz

//struct, abstract class, interface, record yapılanmalarında da metotlar tanımlanmaktadır.

//metotlar bu saydıklarımızın dışında kesinlikle başka bir yerde tanımlanamaz.

//metotlar kesinlikle metotların içerisinde tanımlanamaz demiştik! Yalan!

//C#'da gelen local function özelliği sayesinde metot içerisinde metot tanımlanabilmektedir.

//Erişim belirleyicisi, tanımlanmış metodun tanımlandığı sınıf dışından erişilememesi durumunu belirler.

//Local function'larda erişim belirleyicileri yoktur. Direkt geri dönüş değerinden başlanır.

}

public static int X() //Local function olarak tanımlanan fonksiyon adı, tanımlandığı fonksiyonun adındna farklı olmalıdır! Derleyici hatası vermez! (Ezme durumu olur-Recursive fonksiyonlarla karışır.)

{

Y();

void Y() //X fonksiyonu içerisinde sadece X'ten erişilebilecek bir Y fonksiyonu tanımlamış olduk.

{

System.Console.WriteLine("Merhaba");

//Kullanım Kuralları

//-Bir local function sadece tanımlandığı metodun içerisinde kullanılabilir.

//-Local function'lar tanımlandığı metodun içerisinde her yerden erişilebilir.

}

Y();

return 0;

} } }

- Local function, sadece tek bir metotta tekrarlı bir şekilde kullanılacak bir algoritmayı/kod parçacığını/işlemi o metoda özel bir şekilde tek seferlik tanımlamamızı ve kullanmamızı sağlamaktadır.

- Local function muadilleri: Anonim, Delegate, Func.

**Metotlarda Overloading (Çoklu Yükleme)**

namespace OrnekProje

{

class Program

{

static void Main(string[] args)

{

//Overloading: Çoklu Yükleme

//Bir class içerisinde belirli kurallar çerçevesinde aynı isimde birden fazla metot oluşturmaya

//Method Overloading denir.

//Daha öncede yapmıştık!

//Random random = new Random();

//random.Next(); -> Next(); üzerine geldiğimizde farklı overload'larını görebiliriz.

Matematik matematik = new Matematik();

matematik.Topla(1,2);

//Aynı sayıda parametre ama farklı türde durumu varsa default olarak integer olan fonksiyon tetiklenecektir.

matematik.Topla(5,10,15);

matematik.Topla(5,(byte)2,10,15); //şimdi byte ile tanımlanana gider.

matematik.Topla(1,2,3,4); //int olana gider

matematik.Topla(1,5555555555,3,4);

//doouble olana gider çünkü en yakın değer aralığı karşılayıcısı double. int'in sınırları aşıldı.

matematik.Topla(1,2d,5,23); //double olana gider.

//Kurallar

//Bu metotlar içerisinde fark yaratmamız gerekmektedir. Bu fark bizzat metot imzalarında olmalıdır.

//Metot Overloading işlemini yapabilmek için metot isimleri aynı olmalıdır.

//Metotlar arasında farkı yaratırken erişim belirleyicileri ve geri dönüş değerleri fark yaratmamaktadır.

//Parametre sayıları ya da parametre türleri farklı olmalıdır.

}

}

class Matematik

{

//Bir class içerisinde aynı isimde birden fazla metot tanımlanamaz.

//...

//Belirli kurallar dışında!

public int Topla (int sayi1, int sayi2) //Aynı isimde birden fazla metot var. Demek ki overloading yapılmış.

{

return sayi1 + sayi2; //iki tane parametre alan "topla"

}

private int Topla (int sayi1, double sayi2) //sayi2 farklı türde.

{

return (int)(sayi1 + sayi2);

}

public int Topla(int sayi1, int sayi2, int sayi3) //sayi3'ü eklediğimizde hata ortadan kalkar.

{

return sayi1 + sayi2 + sayi3;//üç tane parametre alan "topla"

}

public int Topla(int sayi1, int sayi2, int sayi3, int sayi4)

{

return sayi1 + sayi2 + sayi3 + sayi4;//dört tane int parametre alan "topla"

}

public int Topla(int sayi1, byte sayi2, int sayi3, int sayi4)

{

return sayi1 + sayi2 + sayi3 + sayi4;//dört tane parametre alan biri de btye olan "topla"

}

public int Topla(int sayi1, double sayi2, int sayi3, int sayi4) //dört tane parametre alan biri de double olan "topla"

{

return (int)(sayi1 + sayi2 + sayi3 + sayi4);

}

//Topla fonksiyonları birbirlerinin overload'u dur.

//Matematik sınıfının altında Topla isminde bir tane fonksiyonum olacak ve bu fonksiyonun da üç tane overload'u olacaktır.

}

}

- Overloading işlemine tabi tutulmuş metotlardan istediğimizi kullanabilmek için o metodun imzasına uygun parametreleri tetiklememiz yeterli olacaktır.

**Recursive(Tekrarlamalı/Özyinelemeli) Metotlar**

- Kendi içerisinde kendisini tetikleyen/çağıran fonksiyonlardır.

- Recursive Fonksiyonlar, bir yaklaşımdır. Anlaşılması, kullanılması ve anlatılması zordur!

- Öngörülemeyen, derinliği tahmin edilemeyen, sonu bilinmeyen durumlarda tercih edilebilir.

Örnek:

X();

void X()

{

System.Console.WriteLine("Merhaba");

X();

}

//Sonsuza kadar ekrana Merhaba yazar.

Örnek:

X();

void X()

{

System.Console.WriteLine("Merhaba");

X();

System.Console.WriteLine("Dünya");

}

//Dünya'yı yazdırmaz. Yine sonsuza kadar Merhaba yazdırır.

Örnek:

//Recursive Fonksiyonlarda kontrolü parametreler ile sağlayabiliriz.

//3 kere Merhaba yazsın istersek;

X();

void X(int a=1)

{

System.Console.WriteLine("Merhaba");

if (a < 3)

{

X(++a); //++a öncelik olarak a ' yı artırıyor ve X'e parametreyi artmış olarak gönderiyor a++ öncelik olarak a ' yı X'e gönderiyor ve a hiç artmadan tekrar X method'u

//çalışıyor ve sonsuz döngü oluyor.

}

}

Çıktı:

Merhaba

Merhaba

Merhaba

Örnek:

//3 kere Merhaba 3 kere Dünya yazalım.

X();

void X(int a = 1)

{

System.Console.WriteLine("Merhaba");

if (a < 3)

{

X(++a);

}

System.Console.WriteLine("Dünya");

}

Çıktı:

Merhaba

Merhaba

Merhaba

Dünya

Dünya

Dünya

Örnek:

//Döngülerin kullanıldığı her noktada recursive fonksiyonlar kullanılabilir. Ama terse her zaman geçerli değildir.

//Belirli değer aralığındaki 5'in katı olan tüm sayıları toplayan recursive fonksiyon'u yazalım.

System.Console.WriteLine(Topla(10,20));

int Topla(int baslangic, int bitis)

{

if(baslangic % 5 == 0 )

return baslangic + Topla(++baslangic, bitis);

if(baslangic < bitis)

return Topla(++baslangic, bitis);

return 0; //toplama işleminde etkisiz eleman.

}

Çıktı:

45

Örnek:

//Bir klasör dizisindeki bütün dosyaları getirelim.

using System;

using System.Collections;

using System.Collections.Generic;

using System.IO;

using System.Linq;

List<FileInfo> files = DosyaYazdir("D:/İndirilenler/Video");

foreach (FileInfo file in files)

{

System.Console.WriteLine(file.FullName); //bunları da foreach ile ekrana yazdıralım.

}

List<FileInfo> DosyaYazdir(string path) //hangi klasörlerini araştırdığımızı path'den öğrenelim

{

List<FileInfo> fileInfos = new();

DirectoryInfo directoryInfo = new(path); //DirectoryInfo ile bütün klasör bilgilerini alalım.

DirectoryInfo[] directoryInfos = directoryInfo.GetDirectories(); //GetDirectories dosyaların içerisindeki directory'leri döner.

//DirectoryInfo dizisi ile bütün directoryInfo'ları aldık.

//hangi dosyanın içerisinde kaç klasör var bilmiyoruz, o yüzden recursive kullanmak lazım.

if(directoryInfos.Any()) //bu directory içerisinde başka directory var mı? Başka bir klasör varsa bunu bilmek için if'ledik.

foreach(DirectoryInfo directory in directoryInfos)//eğer varsa foreach ile directoryInfo'larda biz gezin. Eğer ki bu klasörlerin içinde başka klasörler varsa o klasörlere tek tek git.

fileInfos.AddRange(DosyaYazdir(directory.FullName));//Her DosyaYazdir'da recursive olacak bunu yaparken FileInfos'a AddRange diyerek geriye dönen dosyaları da topla.

//ta ki bunların hepsi DosyaYazdir'dan getirilene kadar.

else //dibine kadar gittikten sonra en son bir tanesinin alt klasörü olmayacak.

fileInfos.AddRange(directoryInfo.GetFiles()); //eğer klasör yoksa var olanları topla, bunları List'e ver geriye döndür.

return fileInfos;

}

**DERS-20: ref ve out Keyword’leri**

**Ref Keyword'ü**

- ref keyword'ü, referanstan gelmektedir.

- Referans, OOP kavramıdır. OOP'de nesneler(object) RAM'de HEAP bölgesinde tutulmaktadır.

- OOP'de referanslar = operatörü ile iletişim geçirebilmektedirler. Bir referans, işaretlediği herhangi bir nesneyi

= operatörü sayesinde farklı bir referansa işaretletebilir. Yani referanslarda = operatörü neticesinde herhangi bir verisel/nesnesel türeme söz konusu

olmamakta, işaretlenmiş nesne diğer referans tarafından da işaretlenmektedir.

int a = 5;

int b = a; //Deep Copy, a çoğaltıldı. Artık b'nin de bellekte yeri ve değeri var.

- ref keyword'ü değer türlü değişkenlerde referans operasyonları yapmamızı sağlayan bir keyword'dür.

Değer türlü değişkenlerin referanslarını çağırmamızı/kullanmamızı sağlar. Referans türlü değişkenler gibi çalışmasını sağlar.

int a = 5;

ref int b = ref a; //artık a'nın değeri değil a'nın referansı(kendisi) gidecek. Artık her iki değişken de aynı bellek adresinden besleniyor anlamına gelir.

a'daki ve b'deki değişiklikler ikisini de etkiler.

- a'nın referansını b'nin referansı ile işaretledik.

- Değer türlü değişkenlerde shallow copy yapmamızı sağlar.

Örnek:

int a = 5;

ref int b = ref a;

System.Console.WriteLine(a);

System.Console.WriteLine(b);

a++;

--b;

System.Console.WriteLine(a);

System.Console.WriteLine(b);

a\*=5;

System.Console.WriteLine(b);

b-=10;

System.Console.WriteLine(a);

Çıktı:

5

5

5

5

25

15

Örnek:

int y = 10;//STACK'te (int)y = 10

X(y); //fonksiyona gönderildi

System.Console.WriteLine(y); //buraya geldiğimizde y hala 10.

void X(int a) //(int) a = 10 çünkü y'nin değeri a'ya gönderildi.

{

a = 25; //(int) a = 25 yani 10 değeri ezildi a 25 yapıldı.

}

Çıktı:

10

Fakat ref kullanırsak;

int y = 10; //STACK'te (int)y = 10

X(ref y); //y'nin değerinden ziyade kendisini gönder.

System.Console.WriteLine(y);

void X(ref int a) //a değişkeni artık bir referans işaretliyor. (int) a = y davranışını sergiler.

{

a = 25; //a'nın değeri 25 olduğunda, a da artık y ise...

}

Çıktı:

25

**C#'ta Ref Returns Özelliği Nedir? Ne Amaçla Kullanılmaktadır?**

- ref keyword'ü sadece metotlarda kullanılır ifadesi yanlıştır. Başka kullanım alanları da vardır.

- ref returns özelliği ise sadece metotlarda çalışır.

- Metotlar geriye değer döndürebilen yapılardır. Ayrıca metotlarda nesneler de döndürebilir.

- ret returns özelliği metotlarda değer türlü değişkenlerin referansını geriye döndürmemizi sağlar.

Örnek:

int a = 5; //Stack'te (int) a = 5;

int b = X(a); //Stack'te (int) b = 'de a'nın değeri fonksiyona gitti.

System.Console.WriteLine(a);

System.Console.WriteLine(b);

int X (int y) //5 değeri geldi. Stack'te (int) y = 5 oldu.

{

y = 25; //5'i ezdi yerine 25 değerini assign etti. (int) y = 25

return y; //y'nin değeri 25'i b'ye gönderdi. (int) b = 25 oldu.

}

Çıktı:

5

25

//ref ile kurcalarsak kodu;

int a = 5; //STACK (int) a = 5

int b = X(ref a); // (int) b = ' geriye bir int türünde referans döndürülmesini bekliyor. y'nin kendisi geriye döner ve (int) b = 25 olur.

System.Console.WriteLine(a);

System.Console.WriteLine(b);

ref int X (ref int y) //a'nın kendisini gönderdik. (int) y = a oldu. y artık a'dır.

{

y = 25; // y'ye 25 değerini atamak demek, a'ya atamak demektir. (int) a = 5 ezildi (int) a = 25 oldu.

return ref y; // y değişkenimizin kendisini geriye döndürdük.

}

//a'da b'de y'de 25'i gösterir.

Çıktı:

25

25

**C#'ta out Keyword'ü Nedir? Ne Amaçla Kullanılmaktadır?**

- out keyword'ü metotların parametreleri üzerinden dışarıya değer göndermenizi sağlayan bir keyword'dür.

- Bir metodun parametreleri varsayılan olarak INPUT'tur. Haliyle metotlarda tanımlanmış parametreler direkt olarak içeriye değer almaya odaklanır.

- Eğer ki bir metodun parametresi dışarıya değer çıkaracaksa o parametrenin out keyword'ü ile işaretlenmesi gerekir.

Örnek:

//bir metot out parametreler barındırıyorsa o parametrelere kendi içerisinde deer atanması gerekmektedir.

//Aksi taktirde derleyici hatası alınacaktır.

//Output parametre barındıran bir metodu kullanırken, out parametrelerden gelecek olan değerleri karşılayacak değişkenler tanımlanmalıdır.

Kullanım 1:

int \_b = 0;

string \_d ="";

int a = X(out \_b, 123, out \_d);

int X(out int b, int c, out string d)

{

b = 24;

d ="a";

return 0;

}

Kullanım 2:

int a = X(out int \_b, 123, out string \_d);

int X(out int b, int c, out string d)

{

b = 24;

d ="a";

return 0;

}

Çıktı: //iki kodda da aynı.

24

ahmet

EKSTRA: TryParse Fonksiyonu

- Hatayı runtime'a bırakmadan, gelen değer kesinlikle integer'a dönüştürülebilir bir değerse eğer bu işlemi yap ve kullan, dememizi sağlar.

string a = "123";

//int.TryParse(a, out int r); a değeri integer'a uygunsa bunu içeride dönüştürüp r ile geri döndür. Ayrıca fonksiyon olarak bool değer de döner.

//bu sayede if'e de uygundur.

if(int.TryParse(a, out int r))

{

}

else

{

}