**Nesne Tabanlı Programlama #1 - Nesne Anatomisi - Referans Türlü Değişkenler**

**Giriş**

**Yaklaşım**

* Duvar örme sanatında bile hangi yaklaşıma göre örüyorsam benim yaklaşımım odur
* Yaklaşım işi yapma ahlakındır. İşi düzenleme, işin matematiğinde hangi formülü kullanmandır.
* Bir yapıya, bir olguya, bir inşaya yaklaşma biçimimizdir. Bir işi hangi felsefeyle ele aldığını anlatır sana yaklaşım.

**OOP**

* Gerçek hayatı, programlama için simüle eden nesneleri baz alan bir programlama tekniğidir
* Yazılım geliştirme süreçlerini oldukça kısaltan ve sistematik hale getiren bir tekniktir.
* Herşey bir nesnedir.
* Gerçek bir sistem, nesnel parçalara ayrılır ve bu parçalar(nesneler) arasında ilişkiler kurulur

**Nesnenin Anatomisi**

* Tüm nesneler esasında bir sınıf modellemesinin örneğidir.
* Kimlik kartının bir modeli vardır.Class modeli 80 milyonda olan nesnesi vardır.
* Bir nesnenin nasıl olması gerektiğine dair bir tane tanımlama/modelleme yapıyorsun ve bundan sınırsız/n adet/belleğinin karşıladığı kadar nesne üretebiliyorsun.

**Nesne Kavramı**

* Nesnellik felsefesine dayanır.
* Gerçek objelerin modellemesi/muadili olarak tanımlama
* Yazılım simülasyonunda sana gerçek hayatıun ta kendisini, karşılığını sunuyor.

**Nesne Modellemesi**

* Nesnelerin oluşturulabilmesi için öncelikle modellenmesi gerekmektedir.
* Nesne içinde bir veya birden fazla değer barındıran değerler bütünüdür.
* Developer Stack'teki değişkene direkt erişebilir ama Heap'e erişim sağlayamaz.Ama Stack'te Heap belleğe erişim sağlayabilir.
* Biz Stack'te Heap'teki nesneleri işaret eden referanslar tanımlıyoruz. Haliyle Stacke'teki refaranslara ben erişiyorum o referansta kendisi Heap'teki nesneye eriştiği için dolaylı yoldan Heap'teki herahangi bir nesneyi işaretlemiş/Erişmiş oluyoruz.

**Sınıf Nedir**

* OOP'de bir object oluşturabilmek için öncelikle o objectin modellemesi/tanımlanması gerekmektedir.
* Bir objenin modelini/tanımını oluşturabilmek için class yapısı kullanılır.
* Class esasında bir nesnenin modelidir.
* Sınıf kodu yazdığımız/inşa ettiğimiz yerdir. Buradaki yazan kod işlevselliğini göstersin dediğimde obje oluşturup bunu objede çalıştırıyoruz.
* Class OOP'nin temelidir.
* Object oluşturabilmek için class'a ihtiyacımız var class üzerinden modelleme yaparız.

**Nesne Tabanlı Programlama #2 - Class Kavramı**

**Sınıf ile Nesne Arasındaki İlişki**

* Sınıf nesne modelidir.
* Object oluşturmadan önce modelleme ve tanımlanmalıdır.
* Objectlerin modelini oluşturmak için classlardan yararlanmaktadır.
* Bu nesne modelinden üretilen nesnelerde operasyon gerçekleştirebiliriz.
* Class'larda nesnelerdeki ortak alan tanımları yapılır.
* Class tekildir. Class'ı bir modele karşı sadece bir kere tanımlarsın. Nesne ondan çoğul olarak türer
* Bire çok bir ilişki vardır.

**Sınıf Nasıl ve Nerede Oluşturulur**

* Class nesnelerindeki ortak alan tanımlaması yapılır.
* Class isimleri tekil olmalıdır çünkü tekil olarak üretilmektedir.
* Class [isim] { } şekliden yapılanması oluşturulur. Class referans türlü bir değişkendir.
* Namespace(İsim Uzayı): İçerisinde birden fazla class struct interface gibi yapılanmaları barındıran esasında senin kütüphane mantığını oluşturmanı sağlayan genel anlamda kurmuş olduğun sistemde sınıflarını farklı namespaceler altında kategorize edip çağırılma esnasında o kategorizeler üzerinden çağırmanı sağlayan bir yapılanmadır.
* Sınıf 3 şekilde oluşturulmaktadır;

1. Namespace İçerisinde :
2. Namespace Dışarısında
3. Class İçerisinde(Nested Type)

* Bir class tanımlanmasında tanımlanan yerde(namespace/dışı,class) aynı isimde birden fazla class tanımlanamaz!
* Aynı namespace altında -> Aynı evde mantığı
* Farklı namespace altında -> Farklı evlerde mantığı
* Namespace üstünde -> Dışarıda mantığı : isim belirtmeksizin erişim.

**Sınıf İle Nesne Modeli Tasarlama**

* !!! İhtiyaçlar !!!
* Class ın içerisinde oluşturulan değişkenlere field denmektedir.
* Classların altında methodlar oluşturulabilir.
* Değerler tutman gerekebilir field. Alan koymam lazım
* Operasyonel/eylemsel işlemler yapmam gerekebilir. Yani fonksiyonelite kazandırmam gerekebilir.

**Sınıf Modelinden Referans Noktası Oluşturma**

* Bir class tanımlandığında o class adı bir türdür. Haliyle o türü kullanabilmek için direkt olarak class adını kullanmamız yeterlidir.
* OrnekModel w; //Senin bunu değişken türü olarak kullanabilmenin altında yatan şey esasında bu bir referans türlü bir değişken olmasıdır.
* Stack'te değişkenlerin değerleri,kendileri ve referans türlü değişkenlerin referansları kendileri tutulur
* Heap'te sadece nesneler tutulur.
* Developer direkt Stack'e erişebilir.Developerların heap'e direkt erişimi engellenmiş Heap'e erişebilmek için bir aracı kullanırız ve aracıyı Stack'te tutarız
* Referans: Stack’te tutulan ve Heap'teki herhangi bir nesneyi işaretleyebilme özelliğine sahip olan değişken türüdür. Biz buna değişken türü değil referans noktası diyoruz. Çünkü referans edebilme özelliğine sahip.
* Referans türlü değişkenler özünde nullable olan değişkenlerdir.
* Referans türünde değişkenler herhangi bir nesne tarafından referans edilmiyorsa nullable olmaktadır.

**Nesne Tabanlı Programlama #3 - Class Members #Field #Property #Encapsulation #Methot #Indexer**

**Class Members(Sınıf Elemanları)**

**Field(Değişken)**

* Nesne içerisinde değer tutmamızı/depolamamızı sağlayan alanlardır.
* Field sadece ve sadece class'ın scope'unda tanımlanır...
* Fieldlar türüne özgü varsayılan değer alırlar.

MyClass m1 = new MyClass(); // Nesne oluşturduk.

* MyClass m1 : Stack'teki m1 referansıyla Heap'teki nesneyi işaretleyecek new MyClass() : Heap'e gidecek nesne

**Property(Özellik)**

* Property esasında özünde bir metottur. Yani programatik/algoritmik kodlarımızı inşa ettiğimiz bir metot.
* Lakin fiziksel olarak metottan farkı parametre almamakta ve içerisinde get ve set olmak üzere iki adet blok almaktadır.
* Biz yazılımcılar nesnelerimiz içerisindeki field'lara direkt erişilmesini istemeyiz.
* Dolayısıyla field'lar da ki verileri kontrollü bir şekilde dışarıya açmak isteriz
* İşte böyle bir durumda metotları kulanabiliriz
* Encapsulation(Kapsülleme/Sarmalama) : bir nesne içerisindeki dataların(field'lardaki verinin) dışarıya kontrollü bir şekilde açılması ve kontrollü bir şekilde veri almasıdır.
* Fieldlar senin cüzdanın sen cüzdanına direkt erişim vermezsin arada sen bulunursun birisi cüzdanına erişmek isterse izin vermezsin arada sen bulunur ve istediği şeyi ister verirsin ister vermezsin.
* Property fieldın gerçek değerini koruyor/kapsüllüyor/sarmalıyor.

MyClass myClass = new MyClass();

Console.WriteLine(myClass.Yasi);

myClass.Yasi = 65;

Console.WriteLine(myClass.Yasi);

* Property'le metot arasındaki farkalrdan bir taneside her property'nin bir türü olmalıdır.Ama metotlar bir geri dönüş türüne her zaman sahip olmayabilirler. Yani void olabilirler.

1.Full Property

Full Property

//Property hangi türden bir field'ı temsil ediyorsa o türden olmalıdır...

//Propertyler temsil ettikleri fieldlar'ın isimlerinin baş harfi büyük olacak şekilde isimlendirilir..

public int Yasi

{

get

{

//Property üzerinden değer talep edildiğinde bu blok tetiklenir.

//Yani değer buradan gönderilir.

return yasi;

}

set

{

//Property üzerine değer gönderildiğinde bu blok tetiklenir.

//Yani değer buradan alınır ve field'a atanır.

//value keywordü Property'nin türü neyse o türe bürünür ve dışarıdan gelen değeri yakalar.

yasi = value;

}

}

Full Property Example

class Banka

{

int bakiye;

string deger;

string deger2;

public int Bakiye //Write & Read Property

{

get

{

if (bakiye > 0)

return bakiye \* 10 / 100;

return 5;

}

set

{

if (value < 10)

bakiye = value;

else

bakiye = value \* 95 / 100;

}

}

public string Deger //Readonly Property

{

get

{

return deger;

}

//set

//{

// deger = value;

//}

}

public string Deger2 //Writeonly Property

{

//get

//{

// return deger;

//}

set

{

deger = value;

}

}

}

1. Prop Property

Prop Property

//readonly olabilir ama writeonly olamaz.

//public int Yasi2

//{

// get

// {

// return yasi2;

// }

// set

// {

// yasi2 = value;

// }

//}

public int Yasi2 { get; set; }//burada field tanımalmaya gerek yok arka planda compiler bunun adına bir field tanımlası yapar.

Auto Property Initializers (C# 6.0)

//Bir property'nin ilk değerini nesne ayağa kaldırılır kaldırılmaz verebiliriz.

//Bu özellik sayesinde read only olan prop'lara hızlıca değer atanabilmektedir.

public int Yasi3 { get; set; } = 15; //Oluşacak değerin/fieldın ilk değeri 15 olsun demek

1. Ref Readonly Returns

Ref Readonly Returns

//ref readonly returns, bir sınıf(class) içerisindeki field'ı referansıyla döndürmemizi sağlayan ve bir yandan da bu değişkenin değerini read only yapan özelliktir.

string adi = "Musa UYUMAZ";

public ref readonly string Adi => ref adi; // bu şekilde field'ın değerine değil fieldın ta kendisine erişmiş olacağız.

1. Computed(Hesaplanmış) Properties

Computed(Hesaplanmış) Properties

//İçerisinde türetilmiş bir bağıntı taşıyan property'lerdir.

int s1 = 5;

int s2 = 10;

public int Topla

{

get

{

return s1 + s2;

}

}

1. Expression-Bodied Property

Expression-Bodied Property

//Tanımlanan property'de Lambda Expression kullanmamızı sağlayan söz dizimidir.

//Sadece readonly property'lerde kullanılır.

public string Cinsiyet => "Erkek";

1. Init Only Properties - Init Accessor (C# 9.0)

Init Only Properties - Init Accessor (C# 9.0)

//Init-Only Properties, nesnenin sadece ilk oluşturulma anında propertylerine değer atamaktadır.

//Böylece iş kuralı gereği run time'da değeri değişmemesi gereken nesneler için bir önlem alınmaktadır.

**Metot**

* Nesne üzerinde, field'larda ki yahut dışarıdan parametreler eşliğinde gelen değerler üzerinde işlemler yapmamızı sağlayan temel programatik parçalardır.
* Property'le metot arasındaki farkalrdan bir taneside her property'nin bir türü olmalıdır.Ama metotlar bir geri dönüş türüne her zaman sahip olmayabilirler. Yani void olabilirler.

MyClass myClass1 = new MyClass();

myClass1.X();

public int X()

{

return 0;

}

**Indexer**

* Nesneye indexer özelliği kazandıran, fıtrat olarak property ile birebir aynı olan elemandır.

public int this[int a]

{

get

{

return a;

}

set

{

a = value;

}

}

**Nesne Tabanlı Programlama #4 - Class Yapısına Dair Son Dokunuşlar**

**Class İçerisinde Tanımlanan Class Sınıf Elemanı mıdır?**

* Bir class'ın içinde bulunan classlar(nested class) Class'ın elemanı olamaz.

**Class Elemanlarına Açıklama Satırı Nasıl Eklenir?**

* Sınıflarıma ve elemanlarına açıklamalar ekleyebiliriz.Bizim sınıfın referansını kullanan herkes bu açıklamalara erişebilir. Bizler bunun için 3 adet (///) slash kullanırız.
* Açıklama satırlarını ne amaçla kullanırız ? : Bizim kendi adımıza yazdığımız kodları daha sonraki süreçlerde geriye dönük baktığımızda daha hızlı kavrayabilmek bu kodu ne amaçla yazdığımızı anlayabilmemiz için oluşturulması gereken notlardır. Nihayetinde kodun izahiyetini/açıklamamızı yapmamızı sağlayan yapılardır.
* Kendi yazdığın kodlar için kendine hatırlatıcı bir unsur olacaktır hem de asıl önemlisi olan senin dışında senden sonra göreve devam edecek olan ya da sisteme daha sonradan dahil olacak olan herhangi bir developer'ın ne olduğuna dair bu kodun açıklanması sistemdeki kodların ne olduğuna dair hızlıca anlamasını sağlayacak bir dökümantasyon görevi gören yapılardır.

/// <summary>

/// Bu bir örnek classtır

/// </summary>

class MyClass

{

int a;

/// <summary>

/// Bu bir property'dir

/// </summary>

public int MyProperty { get; set; }

/// <summary>

/// Bu bir metottur

/// </summary>

public void X() { }

/// <summary>

/// Bu bir örnek metot overloadıdır.

/// </summary>

/// <param name="a">a parametresi...</param>

public void X(int a) { }

public int this[int a]

{

get { return 0; }

}

public class MyClass2

{

}

}

**Nesne Tabanlı Programlama #5 - this Keyword'ü Nedir İşlevleri Nelerdir**

**This Keywordü**

* Bir sınıftan oluşturduğum nesneyi sınıfın modellemesi esnasında temsil etmemi sağlayan keyword this keywordüdür.
* Modelleme de oluşturulan this keywordü her oluşturulan nesne için geçerlidir! benim yorumum
* O anki nesne neyse bu oluşturulmuş olan tüm nesnelerin her birini nesne üzerinde daha doğrusu nesne modellemesi üstünde temsil etmeni sağlayan keyword this keyword'üdür.
* static yapılanmalarda this keywordune erişemeyiz

**1. Sınıfın Nesnesini Temsil Eder**

* Genellikle bu amaçla kullanılır

**2. Aynı İsimde Field İle Metot Parametrelerini Ayırmak İçin Kullanılır**

* eğer bir field ismi ile bir metot parametresinin ismi aynı olursa biz o anki nesnenin temsilcisi olan this'e gidip oradan field'a erişiriz
* this keywordü ilgili class yapılanmasının o anki nesnesine karşılık gelir.
* this kullanmak zorunda değiliz sen burada this kullansanda kullanmasanda compiler seviyesinde otomatik olarak this kullanılmaktadır.

**3. Bir Constructer'dan Başka bir Constructer'ı Çağırmak İçin Kullanılır**

class MyClass

{

int a;

public void X(int a)

{

//bir fieldla aynı isimde parametre olsa bile yakın olana yani metot parametresine erişim sağlar...

this.a = a;

//this.X();

}

}

**Nesne Tabanlı Programlama #6 - Nesne Kavramı Nedir Nesne Nasıl Oluşturulur new Operatörü**

**Nesne Nedir?Ne Amaçla Kullanılır?**

* Nesne dediğimiz yapı canlı bir organizma
* İçerisinde birden fazla anlamlı birbirleriyle ilişkisel veriler tutan ve bunları tutmakla yetinmeyen biraz da entelektüel alışkanlığı/yanı olan bu veriler üzerinde işlemler yapıp sonuçlar üretebilen fonksiyonellikler barındıran bir yapılanmadır/organizmadır.
* Nesne bir veri bütünüdür.
* Bir olguya dair bütün alakadar değerler bir araya geldiğinde nesne ortaya çıkar.
* Class yapılanmalarından üretilen verilerdir.
* Nesneler kompleks verilerdir.
* Nesne dediğin içerisinde lüzumsuz bir değere yer vermeyen yapılanmadır/değerdir.
* Nesneleri modellememizi sağlayan classlar ise Complex Type'dır
* Nesne kodu daha hızlı geliştirebilmek, daha sistematik hale getirebilmek, daha yönetilebilir, daha kullanılabilir kılmak için kullanılır.
* Nesnenin fıtratı class'tır. Nesne oluşturacaksan class'tan başka şansın yok!!!

**Sınıftan Nesne Üretme/Türetme/Oluşturma**

* Nesne üretimi için new operatörü kullanılmaktadır.
* new yazdıktan sonra hangi sınıftan/türden bir nesne oluşturmak istiyorsan ilgili sınıfın ismini bildirmen yeterli olacaktır.
* new operatörüyle herhangi bir türdeki bir sınıftan,ya da herhangi bir sınıftan ya da herahngi bir türden nesne oluşturacaksan bunu yaparken buradaki fonksiyonu çağırmak zorundasın.
* Elimizdeki nesneleri yani Heap'te oluşturmuş olduğumuz o organizmaları/verileri erişip üzerinde işlem yapıp değerleri üzerinde çalışmalar sergilemek istiyorsak bizim bu nesneyi stack'teki bir referansla işaretlememiz lazım.
* Heap'teki nesneyi işaretleyebilmemiz için Heap'teki nesnenin tipi neyse referansının da tipi aynı olmalıdır.Yani biz Stack'te bir referans noktası/değişken tanımladık bu değişkenle Heap'teki/uzaktaki başka bir ortamdaki nesneyi referans etmiş/işaretlemiş olduk.

new MyClass();//new operatörünü gören Compiler : Aha bir tane nesne oluşturacam :):):)

new Random();//Bu bir nesne oluşturma talebidir.

new MyClass();

new MyClass();

new MyClass();

new MyClass();

**Target-Typed New Expressions (C# 9.0)**

* Nesne oluşum sürecinde, oluşturulacak olan nesne eğer ki direkt bir referansa atılıyorsa eğer burada hangi nesnenin oluşturulduğu referans sayesinde bilinebilmektedir. Dolayısıyla ilgili nesnenin oluşturulması için

MyClass x = new MyClass();

//semantiğinden ziyade

MyClass x1 = new();//new dedik

//şeklinde de oluşturmamızı sağlayan yeni bir özelliktir.

class MyClass

{

public int A { get; set; }

public void X()

{

}

}

**Nesne Tabanlı Programlama #7 - Referans Nesne İlişkisi**

**Referans Nedir?**

* RAM'in Stack bölgesinde tanımlanan ve Heap bölgesindeki nesneleri işaretleyen/referans eden değişkenlerdir/noktalardır
* Stack'de tüm değer türlü değişkenlerimi ve değerlerini tutabiliyorum amma velakin referans türl değerlere geçtiğimizde değerler heap'te tutulurken Stack'te referanslar tutulur. Değişkenin kendisi Stack'te karşılığı olacak olan değer Heap'te tutulur
* Interface'ler/Abstract Classlar referans türlü değişkenlerdir. Referans noktası alınabilir değişkenlerdir.
* Referanslar illa bir nesne referans etmek zorunda değildirler.
* Eğer ki bir referans herhangi bir nesne işaretlemiyorsa null değerini alır.

**Stack Heap İlişkisi**

**Nesneler Neden Referansla İşaretleniyor? Neden referans kullanıyoruz?**

* Developer olarak bilgisayarda sabit diske direkt erişebiliyorum,belirli yetkilerim varsa register denilen yapıya erişebiliyorum, bellekte Stack'e erişebiliyorum, Static'e erişebiliyorum amma velakin Heap'e erişemiyorum
* Heap'teki herhangi bir değere/nesneye erişmek için Heap'e dolaylı yoldan erişmem gerekecek haliyle Heap'e erişebilen başka ve benim de erişebildiğim başka bir yapılanmayı kullanmam gerekecek. İşte burada Stack devreye giriyor.
* Biz Stack'e erişebiliyoruz, Stack'teki referanslarda Heap'teki nesnelere erişebildiğinden dolayı biz de dolaylı yoldan Stack üzerinden Heap'teki nesnelerimize erişebilmekteyiz.
* Bundan dolayı nesneleri bir referansla işaretliyoruz.Eğer ki bir nesneyi referansla işaretlemezsek ilgili nesneyi kullanamayız.

**Referans Üzerinden Nesneye Nasıl Erişilebilir?**

* Stack'te oluşturduğumuz değişken sayesinde erişilebilir

**Nesne Üzerindeki Elemanlara Nasıl Erişilir?**

* Elemanlara erişebilmek için yine yapmamız gereken nesneye erişmek (.) operatörü ile dışarıya erişebilirliği olan, dışarıdan erişime izni olan, dışarıdan erişilebilir olan tüm memberlara/elemanlara rahat bir şekilde erişebilmekteyim.

MyClass m = new MyClass();//Burda bir nesne oluşturup m referansıyla işaretlemiş

m.MyProperty = 10;//burda da m referansında bir işlem yapmış

//o referans üzerinden ilgili nesne üzerinde yapmış olduğum herahngi bir işlem daha doğrusu referansın üzerinde yapmış olduğum herhangi bir işlem işaret ettiği nesne üzerinden gerçekleştirilir.

MyClass m2 = null;//Burda m2 referansını oluşturmuş

m2.MyProperty = 20;//Ya kardeşim MyProperty tamam bu türün içinde var ama buna karşılık bir nesne oluşturmadığın sürece bu MyProperty'i nerede çalıştıracağını bilemeyecek yani çalıştıracak bir nesne yok somutlaştırılmamış haliyle program burada patlaycaktır.

* Eğer ki bir referans null ise access modifier(.) operatörü sayesinde ilgili referansın içerisindeki memberlara erişebilirsiniz.Amma velakin referansın bir nesnesi olmadığından dolayı bu member'ı tetikleyecek, bu member'ı üzerinde çalıştıracak bir nesne/ekosistem/organizma olmayacağından dolayı bu referansta null reference hatası verecektir.
* Null olan(yani nesnesi olmayan) referanslar üzerinden herhangi bir member'i çağırıp işlemeye çalıştığımızda bu durumda NullReference hatası verecektir.

**Referanssız Nesneler**

* Her nesnenin bir referansı olmak ya da bir referans tarafından işaretlenmek zorunda değildir!!!
* Bir nesne oluşturulduğu an herahngi bir referansla işaretlenmezse eğer Heap'e yerleştirilir. Lakin bu nesneye tarafımızca bir daha erişemeyiz. Haliyle ilgili nesneyle aramızdaki tek diyalog oluşturma anıdır.
* Heap'te duran nesneler referanssızsa/işaretleyicisi yoksa(Lüzumsuz bir şekilde duran/memory'de bir alan tahsis eden) Garbage Collector mekanizması sayesinde bunlar temizlenecektir.
* Eğer ki bir nesne referanssızsa bunu oluşturabilmekteyiz... Lakin bu nesne sistemde/memoryde lüzumsuz yer kaplayacağından dolayı belli bir süre sonra Garbage Collector dediğimiz çöp toplayıcısı tarafından temizlenecektir.
* Garbage Collector: Heap'de referanssız olan nesneleri imha etmekten/temizlemekten sorumlu bir yapılanmadır.

**Referansız Nesne Üzerindeki Elemanlara Nasıl Erişilir**

new MyClass().MyProperty = 10;

* Bir nesne eğer referanssızsa bu nesneyi tanımladıktan sonra bir daha erişemezsiniz. Dolayısıyla referanssız tanımlanmış olan nesneyi erişebilmenin tek yolu tanımlama anıdır. bir kere erişirsin o da son kez selametle demek için :):)

**Referans Türüne Göre Nesne Elemanlarına Erişim**

* Her bir referans bir nesneyi temsil edecek imzaya sahiptir.
* Benim nesneyi referans edebilmek için o nesneyi temsil edebilcek bir fıtratta class'a ihtiyacım var. bu da nesnenin türündeki class'tır.

**Object Initializer İle Nesne Oluşturma Esnasında Propertylere İlk Değer Atama**

* sadece field ve propertylere değer atama yapılabilir methodlara atama yapılamaz ve çağırılamaz

MyClass m4 = new MyClass()

{

MyProperty = 10,

MyProperty2 = 30,

MyProperty3 = 40,

};

MyClass m5 = new MyClass();

m5.MyProperty = 10;

#endregion

}

}

class MyClass

{

public int MyProperty { get; set; }

public int MyProperty2 { get; set; }

public int MyProperty3 { get; set; }

}

**Nesne Tabanlı Programlama #8 - Nesne Kopyalama Davranışları Shallow Copy Deep Copy**

**Nesne & Değer Kopyalamadan Kastedilen Nedir?**

* Belleğin stack kısmında tanımlanan ve değerlerini orada tutan değişkenlere Değer türlü değişkenler denir.
* Nesnelerden farkı sadece değersel olmaları/sadece veri olmaları

int a = 5;

int b = a;

**Shallow Copy**

* Var olan bir nesnenin/değerin , referansının kopyalanmasıdır.Shallow copy neticesinde eldeki değer çoğaltılmaz.Sadece birden fazla referansla işaretlenmiş olur.
* Nesne tek lakin işaretleyen referans sayısı birden fazla!
* Bir nesneyi birden fazla referans işaretliyorsa ve bu nesne türetilmiyorsa/çoğlatılmıyorsa biz buna shallow copy deriz.
* Referans türlü değişkenlerin/değerlerin default davranışı shallow copy'dir.

MyClass m1 = new MyClass();

MyClass m2 = m1;

* Şu davranış eğer değer türlü değişkenler üzerinde yapılıyorsa eğer deep copy oluyordu yani değerler çoğaltılıyordu.
* Eğer bu davranış referans türlü değişkenlerde oluyorsa burada shallow copy yapılır.Yani ilgili nesne ilgili değer türetilmez/çoğaltılmaz sadece işaretlenir.
* Biz artık her iki referanstan'da aynı nesneyi elde ederiz.
* Burada yüzeysel/sığ bir kopyalama yapılıyor.

MyClass m3 = m2;

MyClass m4 = new MyClass();

MyClass m5 = null;

MyClass m6 = new MyClass();

MyClass m7 = m6;

m5 = m7;

MyClass m11 = new MyClass();

MyClass m12 = new MyClass();

MyClass m13 = m11;

m11 = m12;

m12 = m11;

m11 = m11;

* Bir referans sadece ve sadece tek bir nesneyi işaretler.Birden fazla nesneyi işaretlemesi mümkün değil.
* Bir referans önceden bir nesneyi işaret ediyorsa o referansa başka bir nesneyi işaret etmesini söylediğin anda önceki işaretleme/referans kopacaktır.

**Deep Copy**

* Değer türlü değişkenlerde default davranış deep copy'dir.
* Bir metodun parametresinde ya da bir yerde bir değişkenene ihtiyacın olup gönderdiğin zaman oraya değişkenin kendisi gitmez değişkenin değeri kopyalanır hafızada ve bu değer gönderilir.
* Davranış olark derin kopyalama yapmış oluyorsun Kopyalandıktan sonra oluşan değerler birbirinden bağımsız, aynı fıtratta olan bağımsız değerler/nesneler olacaktır.

MyClass m111 = new MyClass();

MyClass m112 = m111;//Shallow Copy

MyClass m113 = m111.Clone();//Deep

//m112 üzerinde değişiklik yaparsan m111 etkilenecek, m111 üzerinde değişiklik yaparsan m112 etkilenecek ama m113 üzerinde ne değişiklik yaparsan yap her ikiside etkilenmeyecek.Çünkü artık bu klonlanmış darklı bir değer/nesne bu

}

}

class MyClass

{

public MyClass Clone()

{

return (MyClass)this.MemberwiseClone();//MemeberwiseClone bir sınıfın içerisinde o sınıftan üretilmiş olan o anki nesneyi clonelamamızı sağlayan bir fonksiyondur...

}

}

**Nesne Tabanlı Programlama #9 - Derinlemesine Encapsulation Nedir**

**Encapsulation nedir? Bir Veriyi Neden Kapsülleriz?**

* Encapsulation, nesnelerimizde ki field'ların kontrollü bir şekilde dışarıya açılmasıdır.
* Bir başka deyişle, nesnelerimizi başkalarının yanlış kullanımlarından korumak için kontrolsüz değişime kapatmaktır.
* Nihayetinde ben bir nesne oluşturuyorum oluşturmuş olduğum nesneyi dışarıdan kullanacak bir adam tarafından benim dışında yanlış kullanılmasını istemiyorsam eğer ilgili nesnenin fieldlarını kapsüllemem lazım.
* Encapsulation dediğimiz yapılanma fieldların kontrolünü sağlamaktır.
* Encapsulation ise tam tersi. Encapsulation, motor üreticilerinin onu kaporta ve motor kapakları ile koruma içgüdüsüdür. Encapsulation evlerimizde ki kapı ve perdelerdir. Encapsulation beynimizi çevreleyen kafatasıdır. Encapsulation, sınıfımızı başkalarının yanlış kullanımlarımdan korumak için kontrolsüz değişime kapamaktır. Sınıfımızın içeriğini deterministic tutabilmek için koruruz onu.

**Encapsulation Nasıl Uygulanır?**

**Metot İle Encapsulation**

**Eskiden Encapsulation**

MyClass m = new MyClass();

m.ASet(15);

Console.WriteLine(m.AGet());

**Property İle Encapsulation**

m.A = 1412214;

Console.WriteLine(m.A);

class MyClass

{

int a;

#region Eskiden Encapsulation Nasıl Yapılıyordu

public int AGet()//Bu fonksiyon tetiklendiğinde o anki nesnedeki a field'ının değerini geri gönder

{

return this.a;

} //Dış dünyaya açmak istediğin field'ın neyse o türde çalışmalısın

public void ASet(int value)

{

this.a = value;

}

#endregion

public int A

{

get { return a; }

set { a = value; }

}

}

**Nesne Tabanlı Programlama #10 - Record Nedir OOP'de ki Yeri Neresidir (C# 9.0)**

**Record'ı Anlayabilmek İçin Ön Hazırlık)(Init - Only Properties)**

* Herhangi bir nesnenin propertylerine ilk değerlerinin verilmesi ve sonraki süreçte bu değerlerin değiştirilmemesini garanti altına almamızı sağlayan Init - Only Properties özelliği gelmiştir
* Bu özellik sayesinde nesnenin sadece ilk yaratılış anında propertylerine değer atanmakta ve böylece iş kuralları gereği runtime'da değeri değişmemesi gereken nesneler için ideal bir önlem alınmaktadır.
* Init - Only properties developer açısından süreç esnasında değiştirilmemesi gereken property değerlerinin -yanlışlıkla- değiştirilmesinin önüne geçmekte ve böylece olası hata ve bug'lardan yazılımı arındırmaktadır.

**Only - Init Properties VS Getter - Only Properties**

* Bu ikisi arasındaki temel fark esasında Object Initializer işlevselliğinden kaynaklanmaktadır.
* Init-Only Properties özelliğine ‘init’ keyword ‘ü eşlik etmektedir.

Public string Name{**get; init;**}=”Kutsal İsyan”; // Readonly dir değer okunur değiştirilemez.

MyClass my = new MyClass()

{

MyProperty = 25,

A = 5

};

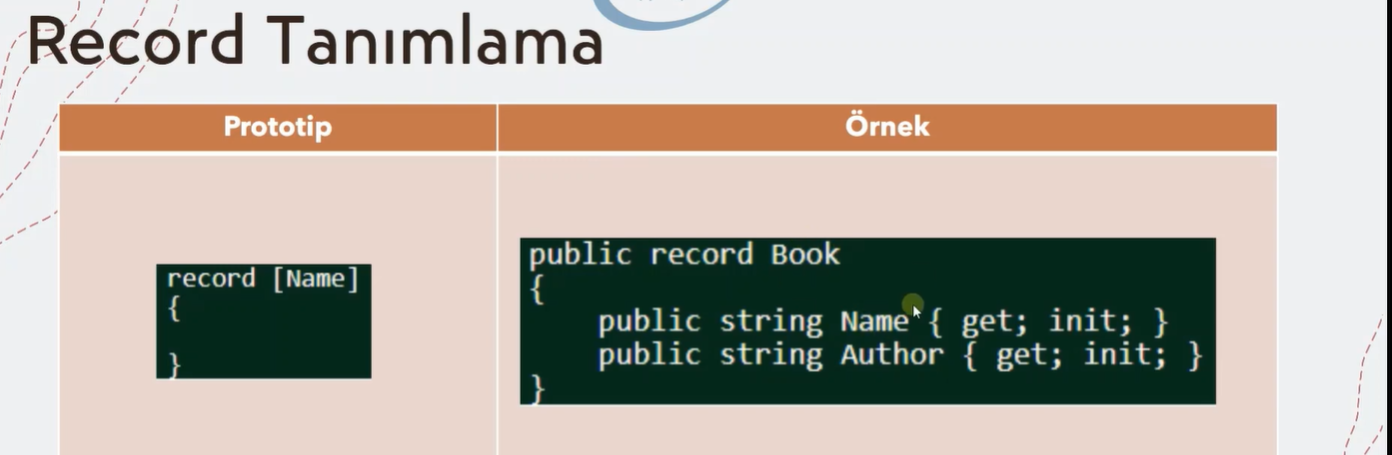
//my.MyProperty = 123;

Console.WriteLine(my.MyProperty);

**Records Nedir?**

* Init-Only Properties özelliği, nesne üretim esnasının dışında değişmez değerler oluşturması için constructor ve auto property initializers yapısının yanında object initializer yapısının kullanılabilir olmasını sağlıyordu.
* Eğer ki tek bir property'de Sabitlik/Değişmemezlik/Salt okunurluk/Readonly'lik/Sadece okunabilirlik amaç ediniliyorsa Init-Only Properties özelliği kullanılır.
* Eğer ki bir objeyi bütünsel olarak değişmez yapmak istiyorsak o zaman initten daha fazlasına ihtiyacımız olacaktır. İşte bu ihtiyaca istinaden Records türü geliştirilmiştir.
* Record bir objenin topyekûn olarak sabit/değişmez olarak kalmasını sağlamakta ve bu durumu güvence altına almaktadır.
* Böylece bu obje artık değeri değişmeyeceğinden dolayı esasında objeden ziyade bir değer gözüyle bakılan bir yapıya dönüşmektedir.
* Recordlar classlara istinaden objeden ziyade içerisinde bulunan dataları sabitleyerek nesneden ziyaden verilerini datalarını öne çıkarmaktadır.
* Recordlar bir classtır. Sadece nesnelerinden ziyade değeri ön plana çıkmış bir classtır.
* Nesne ön plandaysa bu class, nesnenin değerleri ön plandaysa bu record'dur.





MyRecord2 record1 = new MyRecord2() { MyProperty = 5 };

MyRecord2 record2 = new MyRecord2() { MyProperty = 5 };

Console.WriteLine(record1.Equals(record2));

MyClass2 myClass1 = new MyClass2() { MyProperty = 5 };

MyClass2 myClass2 = new MyClass2() { MyProperty = 5 };

Console.WriteLine(myClass1.Equals(myClass2));

//MyClass3 myClass3 = new MyClass3

//{

// MyProperty1 = 5,

// MyProperty2 = 10,

//};

//MyClass3 myClass31 = new MyClass3

//{

// MyProperty1 = myClass3.MyProperty1,

// MyProperty2 = 15

//};

//MyClass3 myClass32 = myClass3.With(24);

MyRecord3 myRecord3 = new MyRecord3

{

MyProperty1 = 5,

MyProperty2 = 10

};

MyRecord3 myRecord31 = myRecord3 with { MyProperty2 = 15 };

}

}

record MyRecord2

{

public int MyProperty { get; init; }

//public void X()

//{

//}

//public MyRecord()

//{

//}

//~MyRecord()

//{

//}

//static MyRecord()

//{

//}

}

class MyClass2

{

public int MyProperty { get; set; }

}

class MyClass

{

readonly int a;

public int A

{

get

{

return a;

}

init

{

a = value;

}

}

public int MyProperty { get; init; } = 3;

}

record MyRecord3

{

public int MyProperty1 { get; init; }

public int MyProperty2 { get; init; }

}

class MyClass3

{

public int MyProperty1 { get; init; }

public int MyProperty2 { get; init; }

public MyClass3 With(int property2)

{

return new MyClass3

{

MyProperty1 = this.MyProperty1,

MyProperty2 = property2

};

}

}

**Nesne Tabanlı Programlama #11 - Özel Sınıf Elemanları - Constructor Metot Nedir**

**Özel Fonksiyonlar**

* Bu sınıfın üzerinden bu sınıftan üretilecek nesnenin üzerinde bu nesnenin üretim esnasındaki yapılacak operasyonları tanımlamamızı sağlayacak aynı şekilde üretilen bu nesne ebedi değildir bir gün imha edilir işte bu imha edilme sürecinde son kez yapılacak işlemlere dair tanımlamaları yapmamızı sağlayacak olan özel fonksiyonlarımız vardır.
* Bu özel memberlar tüm sınıflarda ortak olarak kullanılabilir yapılanmalara sahiptir.

**Construtor Nedir?**

* Biz bir nesne oluştururken aslında bir fonksiyon tetikliyoruz.
* Adı üstünde yapıcı/inşa edici metot Nesnenin yapıcılığını nesne ilk ayağa kaldırılırken o nesneye dair konfigürasyonları yaptığımız metottur.
* Nesne üretilirken ilk tetiklenen metottur.
* Constructor nesne oluşturma sürecinde tetiklenmek zorundadır.

**Constructor Davranış Modeli**

* Constructor new ile nesne yaratma talebi geldikten ve ilgili nesneye hafızada yer ayrıldıktan sonra tetiklenir.

**Constructor Metot Nasıl Oluşturulur?**

* Özel olsada fıtrat olarak metottur.
* Metot adı sınıf adıyla aynı olmalıdır!(Özel sınıf elemanlarının dışında hiçbir member sınıf adıyla aynı olamaz!)
* Geri dönüş değeri olmaz/belirtilmez!
* Erişim belirleyicisi public olmalıdır!(private olduğu durum ayriyetten incelenecektir.)
* Bir constructor tanımlamak için O sınıfla aynı isimde ve geri dönüş değeri bildirilmemiş olan bir imza atmanız gerekiyor
* Constructor her daim tetiklenmesi gereken zoraki bir fonksiyondur.

new MyClass(5);

new MyClass(10);

new MyClass();

new MyClass();

new MyClass();

MyClass m = new(15);

class MyClass

{

public MyClass()

{

}

public MyClass(int a)

{

Console.WriteLine("Bir adet myclass nesnesi oluşturulmuştur. " + a);

}

public MyClass(string a)

{

}

public MyClass(string a,int b)

{

}

public void X()

{

}

}

**Default Constructor**

* Her sınıfın içerisinde tanımlamasak dahi var olan bir default constructor vardır.
* Eğer ki bir sınıf oluşturdun ve içine herhangi bir constructor koymadın içinde belki hiç birşey yoktur ama compilr seviyesinde default bir constructor gene tetiklenecektir.
* Eğer ki bir class'a constructor eklersek default constructor'ı ezmiş oluruz. Ezdiğimizde de bizim yazdığımız Constructor geçerli olacaktır.

**Parametreli Constructor**

* Constructorlar parametre alabilen yapılardır.

**Constructor Overload**

* Biz bir sınıf içinde birden fazla Constructor metot tanımlayabiliyoruz.
* Bir sınıfın içinde aynı isimde birden fazla member olamaz Haliyle bir sınıf içerisinde sadece ve sadece bir isimde bir tane metot olabilir.
* Eğer ki bir sınıfta birden fazla aynı isimde metot oluşturacaksak bu metotlar overloading yani çoklu yüklenme yapılmış olması lazım. Bir isme çoklu yükleme yapılır
* İmza yapılanması aynı ama bu imzaların nitelikleri değişiyor her metodun ismi aynı olmalı overload olabilmesi için ve parametrelerin sayıları yahut türleri,yahut yerleri değişmiş olmalıdır.

**Constructor'ın Erişim Belirleyicisini private Yaparsak Sorunsalı?**

* Bir sınıfın constructor'ını private yaparsak ilgili sınıfın private yapılan o member'ı dışarıdan nesne üzerinden erişilemeyeceğinden dolayı eğer ki bu member Constructor ise eğer yine erişilemeyecektir ve nesne üretiminde hata alacağız.
* Eğer constructor metodu tetikleyemiyorsanız ilgili sınıftan nesne üretemezsiniz.
* Eğer ki sen bir sınıfın constructor metodunu private yaparsan o sınıfın constructor'ına erişilemeyeceği için nesne üretim esnasınıda baltalamış olursun haliyle nesne üretimini engellemiş olursun.

new MyClass2();

class MyClass2

{

MyClass2()

{

}

void X()

{

new MyClass2();

}

}

**this Keywordüyle Constructor'lar Arası Geçiş**

* Bir sınıfın içinde this keywordü o sınıfın o anki nesnesini temsil eder
* this keywordü nasıl ki sınıfın o anki nesnesini temsil ediyor aynı şekilde this keywordu bir sınıfın nesnesinin içerisindeki birden fazla constructorlar arasında geçiş yapabilme sorumluluğunuda üstlenebilmektedir'larında arasında geçiş yapmasını sağlayabilen bir nitelik taşımaktadır.
* this keywordü constructorlar arasında atlamayı yani constructor'ın herahngi birisinden farklı constructorları tetikleyebilmemizi sağlayan bu sorumluluğu üstlenen bir keyword'dür

new MyClass3();

new MyClass3(5);

new MyClass3(5,10);

class MyClass3

{

public MyClass3()

{

Console.WriteLine("1. Constructor");

}

public MyClass3(int a) : this()//Herhangi bir constructor'ın yanına bu şekilde this keywordünü verirsen o anki constructor'ın dışındaki diğer constructorlara erişmeni sağlayacaktır

{

Console.WriteLine($"2. Constructor : a = {a}");

}//Eğer ki nesne üretiminde bu constructor tetikleniyorsa içerideki işlemleri gerçekleştirmeden önce this() deki constructor'ı tetikleyecektir.

public MyClass3(int a,int b) : this(a)//burada parametre olarak manuel ve sadece çağrılan yerin parametlerine erişim sağlayabilir.

{

Console.WriteLine($"3. Constructor : a = {a} | b = {b}");

}

}

**Record'lar da Constructor**

* Record'ın özünde/esasında sınıftan bir farkı yok bu yüzden constructor için geçerli tüm kurallar burada da geçerlidir.

record MyRecord

{

public MyRecord()

{

}

public MyRecord(int a) : this()

{

}

}

**Nesne Tabanlı Programlama #12 - Özel Sınıf Elemanları - Destructor Metot Nedir**

**Destructor/Finalizer Functions**

* Bir class'tan üretilmiş olan nesne imha edilirken otomatik çağrılan/tetiklenen metottur.
* Nesneyle işimiz bitti.nesneyle ilgili bu değerle ilgili işlem yapmama gerek yok dolayısıyla böyle bir durumda nesneyi yok edeceğiz.çünkü biz belleğimizde bize lazım olmayan bir değerle işimiz olmayacak.
* Nesne imha edilirken/yok edilirken son kez nesne adına selametle dememizi sağlayacak olan bir fonksiyondan bahsediyoruz
* Nesnede ki garanti bri şekilde en son tetiklenecek olan fonksiyon destructor fonksiyonudur.
* C#'ta destructor sadece class yapılanmasında kullnılabilir ve bir class sade ve sadece tek bir destructor içermelidir.
* Destructor parametre alamaz

**Destructor Davranış Modeli**

* Bir nesneyi biz edebiyen kullanmayız nesne ile işimiz bittiğinde yani nesneyi imha ederken son kez hadi hoşçakalın dediği fonksiyona biz destructor diyoruz
* Destructor yıkıcı metot nesne imha edilirken/yıkılırken/yok edilirken/silinirken devreye girecektir.

**Bir Nesne Hangi Şartlarda Kim Tarafından İmha Edilir?**

* Bir nesnenin imha edilebilmesi için;
  1. İlgili nesne herhangi bir referans tarafından işaretlenmemelidir
  2. Yahut nesnenin oluşturlduğu ve kullanıldığı scope sona ermiş olmalıdır.
  3. Yani anlaşılan ilgili nesneye bir daha erişilemez hale gelinmelidir.

**Garbage Collector**

* Ram'de bir daha kullanılmamak üzere/erişilmemek üzere duran nesneler/referanssız nesneler ya da ilgili alanda scope'u bitmiş görevi bitmiş alanlarda tanımlanmış nesneler dışarıdan bir daha erişilmeyen nesneler bunlar çöp diye nitelendirilir.
* Uygulamada lüzumsuz olan nesneleri toplamak için Garbage Collector isimli bir mekanizma devreye girer.
* Esasında Garbage Collector C#'ta bellek optimizasyonunu üstlenen bir yapılanmadır. Yani .Net'te bellek optimizasyonu/bellek yönetimi biz developerlardan alınıp Garbage Collector'a verilmiştir. ve mimari/işletim sistemi bunu otomatik olarak yürütür.
* Bu mekanizma kendi kafasına göre çalışır.
* C#'ta Garbage Collector'ın ne zaman iş göreceği tahmin edilemez.Kafasına göre takılır:)
* Dolayısıyla biz geliştiricilerin bu mekanizmaya müdahale etmesi pek önemli değildir.
* Bu mekanizmaya biz developerların bu mekanizmaya çokta müdahale etmesi tavsiye edilmez
* Nihayetine bizler buna müdahale ettikçe/işine karıştıkça orada büyük bir maliyete sebeb olabilmekteyiz. Garrbage Collector dediğimiz mekanizma bellek optimizasyonunu kendi planına göre yürüttüğünden dolayı müdahale edebiliyoruz ama bu müdahale çok fazla önerilmiyor.

**Destructor Tanımlama Kuralları**

* Nesne tam imha edilirken nesne son kez hadi eyvallah diyeceği destructor fonksiyonunu devreye sokuyor onu tetikliyor.

X();

GC.Collect();//Garbage Collector devreye sokulmuş oldu.

Console.ReadLine();

static void X()

{

MyClass m = new MyClass();

}

class MyClass

{

public MyClass()

{

Console.WriteLine("Nesne üretilmiştir");

}

~MyClass()//İmhadan bir tık öncesi son nefes

{

Console.WriteLine("Nesne imha ediliyor...");

}

}

int sayi = 100;

while (sayi >= 1)

{

new MyClass2(sayi--);

}

Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

GC.Collect();

Console.ReadLine();

class MyClass2

{

int no;

public MyClass2(int no)

{

this.no = no;

Console.WriteLine($"{no}. nesne oluşturulmuştur.");

}

~MyClass2()

{

Console.WriteLine($"{no}. nesne imha ediliyor");

}

}

**Nesne Tabanlı Programlama #13 - Özel Sınıf Elemanları - Deconstruct Metot Nedir**

**Deconstruct Metodu Nedir?**

* Bir sınıf içerisinde deconstruct ismiyle tanımlanan metot compiler tarafından özel olarak algılanmakta ve sınıfın nesnesi üzerinden geriye hızlıca Tuple bir değer döndürmemizi sağlamaktadır.
* Bir sınıfınız olduğunu düşünün bu sınıftan ürettiğiniz bir tane nesne oldğunu varsayın.Şimdi bu nesnenin içerisinde onlarca property'niz olabilir. bu propertylerden belli başlı olanları ihtiyacınıza istinaden belli başlı olanları hızlıca özet bir analiz mahiyetinde elde etmek istiyor olabilirsiniz işte bu ihtiyaca istinaden size bu nesnenin ilgili propertylerini hızlı bir şekilde belirli bir semantikle verebilcek ve bunu tuple olarak verebilcek bir özelliktir.

Person person = new Person

{

Name ="Musa",

Age = 23

};

var (x, y) = person;

class Person

{

public string Name { get; set; }

public int Age { get; set; }

public void Deconstruct(out string name,out int age)

{

name = Name;

age = Age;

}

}

**Nesne Tabanlı Programlama #14 - Özel Sınıf Elemanları - Static Constructor Metot Nedir**

**Static Constructor**

* Bir sınıftan nesne üretilirken constructor'a nazaran ilk tetiklenen metot static constructor'dır. Amaaa...!!!
* Bir sınıftan nesne ğretilirken ilk tetiklenen fonskiyon Static Constructor'dır.... Amma velakin belirli bir duruma istinaden!!!!
* Ctor ilgili sınıftan herrr nesne üretilirken tetiklenen fonksiyondur.
* Static constructor ilgili sınıftan ilk nesne üretilirken bir kereye mahsus tetiklenen metottur.
* Bu sınıftan ilk defa bir nesne üretiliyorsa ilk tetiklenen fonksiyon static constructor'dır ardından constructor'dır. Yok eğer bu sınıftan ilk değil de n. nesne üretiliyorsa artık static constructor bir daha tetiklenmez çünkü uygulamada o sınıftan bir yerlerde bir zamanlarda ilk olan bir nesne üretilmiştir. O zaman tetiklenmiştir.
* Static yapılanmalar uygulama bazlı datalarımızı yerleştirdiğimiz alandır.
* Static diye bişey duyduğunuzda nesneden bağımsızdır.Bir kereye mahsus tetiklenir.Bir değer koyarsın her yerde aynı değer söz konusudur.
* Static constructor'da geri dönüş değeri ve erişim belirleyicisi bildirilmez!
* Overloading yapılmaz! Bir sınıfın içerisinde sade ve sadece bir tane tanımlanabilir. Yani parametre almaz!!
* Static constructor öyle ya da böyle tetiklendiği zaman bir daha tetiklenmez. Bir sınıfta tetiklenme hakkı bir keredir.

new MyClass();

new MyClass();

class MyClass

{

public MyClass()

{

//Bu sınıftan nesne üretilirken ilk tetiklenecek olan metottur

Console.WriteLine("MyClass constructor'ı tetiklenmiştir.");

}

static MyClass()

{

//Bu sınıftan ilk nesne üretilirken ilk tetiklenecek olan metottur

//Üretilen ilk nesnenin dışında bir daha tetiklenmez

Console.WriteLine("MyClass static constructor'ı tetiklenmiştir.");

//Static constructor'ın tetiklenebilmesi için illa ilk nesne üretimi yapılamsına gerek yoktur. İlgili sınıf içerisinde herhangi bir static yyapılanmanında tetiklenmesi static constructor'ın tetiklenmesini sağlayacaktır.

}

}

**Singleton Design Pattern**

* bir sınıftan uygulama bazında sade ve sadece tek bir nesne oluşturulmasını istiyorsan kullanabileceğin bir design pattern.
* Design pattern yapılanmalar oop nimetlerinden faydalanarak geliştirdiğimiz stratejiler tekrar eden stratejiler yani dünyada her yerde standart olan stratejiler.
* Bu uygulamada bu sınıftan sadece bir tane nesne oluşsun iki tane oluşmasın diyipte bu sınıftan sade ve sadece bir tane nesne olacak ne zaman yeni bir nesne ihtiyacın varsa var olanı sana döndürüp onu kullanmanı sağlayacak bir ihtiyacın varsa bu strateji senin bu ihtiyacına çözüm getirebiliyor.
* Bu dizayn pattern'ı uygularken Static constructor dediğimiz yapılanma kullanılabiliyor.

var database1 = Database.GetInstance;

var database2 = Database.GetInstance;

var database3 = Database.GetInstance;

database1.ConnectionString = "afasgagagawgawg";

class Database

{

Database()

{

}

public string ConnectionString { get; set; }

static Database database;

public static Database GetInstance

{

get

{

return database;

}

}

static Database()

{

database = new Database();

}

}

**Nesne Tabanlı Programlama #15 - Positional Record Nedir**

**Positional Record**

* Aynı şekilde nesnenin birebir ta kendisi olan sadece davranışsal olarak datasını ön planda tutan Record denilen yapılanmaların da kendine has var olan özel memberları özel bir semantikle kullanabilmemizi sağlayan yapılardır.
* Norminal Record'lar Object Initializer'lar ile ilk değerleri verilerek üretilebilen readonly datalardı
* Positional Record'lar ise esasında Record'lar içerisinde tanımlama yapabildiğimiz constructor ve deconstructor kullanımlarını daha da özelleştirerek kullanılmasını sağlamaktadır.
* Positional Record kullanırken parametrelerin karşılıkları olan propertyleri manuel oluşturmak zorunda DEĞİLİZ...
* Positional Record tanımlanmışsa eğer nesne üretiminde tetiklenmesi/kullanılması zorunludur.

MyRecord m = new MyRecord("sgagfag","adfafasf");

var (n, s) = m;

record MyRecord(string Name,string Surname)

{

public MyRecord() : this("agfagagaswg","afagfawfgawfwa")

{

}

public MyRecord(string name): this()

{

}

//public string Name => name;

//public string Surname => surname;

}

**Nesne Tabanlı Programlama #16 - Inheritance(Kalıtım) Nedir 1. Bölüm**

**Inheritance(Kalıtım)**

**Bir Programcı Açısından Kalıtım Nedir?**

* OOP yaklaşımı gerçek hayattan modellenerek yazılıma uygulanmış/uyarlanmış bir yaklaşımdır. Kalıtımı adamlar demiş ki ulan biz oop diye bir yaklaşım geliştiriyoruz bunu daha iyi nasıl yapabiliriz demişler biyolojideki kalıtım mantığını alıp buraya uyarlamışlar
* Nesne tabanlı programlama bir tek yazılımı programlama değil günlük hayatı programlama üzerine de bizlere mantık kazandırır.
* Kalıtım OOP'nin ennnn önemli özelliğidir.
* Üretilen nesneler farklı nesnelere özelliklerini aktarabilmekte ve böylece hiyerarşik bir düzenleme yapılabilmektedir.
* Aynı aile grubundan gelen nesnelerin ya da yatayda eşit seviyede olan tüm olguların benzer özelliklerini tekrar tekrar her birinde tanımlamaktansa bir üst sınıfta tanımlanmasını ve her bir sınıfın bu özellikleri üst sınıftan kalıtımsal olarak almasını sağlamaktadır.
* böylece hem kod maliyeti düşmekte hem de mimarisel tasarım açısından avantaj sağlamaktadır.
* Kod maliyetini düşürür.
* Kalıtım denen olay lazımsa kullanılır.
* Bizim işimiz bir yerden sonra var olan bir şeyi tekrar tekrar yazmak değil Biz durmandan stratejik bir yapılanmaya kayacağız Senin yazacağın kodlar kullanılabilir olmalı lazım olduğu noktada kullanılabilir olmalı eğer kullanamıyorsan tekrardan lazım olduğu noktada yazıyorsan orada bir problem vardır.
* Nesne tabanlı programlamada benzer/aynı ol gudaki nesnelerin aynı olan memberları/özellikleri/içerikleri eğer ki her sınıf içinde tekrar tekrar tanımlanmışsa bu aykırı bir durumdur!
* Aynı olguda olan sınıfların tekrar eden memberları bir başka sınıfta toplansın ve oradan ilgili sınıflara kalıtımsal olarak aktarılsın
* Genellenemeyen, diğerlerinde olmayan ve sadece o sınıfa ait olan özellikler direkt ilgili sınıfta tanımlanmalıdır
* OOP'de kalıtım özünde nesnelerin birbirinden türemesini sağlayan bir özelliktir.
* Bu özellik yanında da birçok özellik ve stratejik yapılanma getirmektedir.

**Kalıtımın Kullanılmadığı Durum**

* Benim sınıflarımın içerikleri aynıysa bir kendine sorarsın benim yazdığım kod ne kadar kaliteli Bir kod yazdık çalışıyor nesne üretebiliyorum mis gibi
* ama benim bir cümle kurma tarzım var bir de o cümleyi üslubuyla tanımlı kurma şansım var kısacık cümleyle derdimi anlatıp çözüm getirmek varken uzun uzun derdimi anlatıp daha sonradan da o derdi komplike hale getirebilecek şekilde çözmek var
* Aynı olguda olan sınıfların tasarımına baktığında tekrar eden yapılanmalar varsa diyeceksin ki ulan burda bir yanlışlık olabilir ben buradaki hepsinde tekrar eden yapılanmaları bir farklı sınıfa alıp oradan kalıtım alabilirim diyeceksin

**Kalıtımın Kullanıldığı Durum**

* Kalıtım lalettayin bir şekilde tasarlanmamalıdır. Ortak olguda olan nesneleri temsil edecek olan bir üst ve daha evrensel nitelikte olgu olmalıdır.Opel,Mecedes ve Fiat ortak olgudur. Yani üçüde bir arabadır. Haliyle bunların daha evrensel üst niteliği Araba olarak nitelendirilir.
* Kalıtım operasyonunda kalıtım veren sınıfın erişilebilen tüm memberları kalıtım alan sınıfa kalıtsal olarak aktarılacaktır.
* Kodları tekrar tekrar tüm sınıflarda yazmaktansa kalıtımdan faydalanmak en doğrusu!
* Tek bir merkezi yerden tekrar eden yani ortak olan yapılanmaları yöenetebiliyorum. kodlarımız
* Hem yönetilebilirliği artmış oldu hem de kod maliyetini düşürmüş olduk.

**C# Programlama Dilinde Hangi Yapılar Kalıtım Alabilirler?**

* Kalıtım sınıflara özel bir niteliktir.
* Yani bir sınıf sade ve sadece bir sınıftan kalıtım alabilir.
* record'lar da kalıtım alabilmekte Lakin sadece kendi aralarında.Kalıtım alabildikleri tek istisnai sınıf Object sınıfıdır
* Ayrıca abstract class, interface ve struct gibi yapılarında kendilerine göre kalıtımsal operasyonları mevcuttur.

**C#'ta Kalıtım Nasıl Alınır? (: Operatörü)**

* İki sınıf arasında kalıtımsal ilişki kurabilmek için : operatörü kullanılmaktadır.
* Hatta bilsekte bilmesekte kalıtımsal tüm ilişkiler : operatörü tarafından yapılmaktadır
* class Opel : Araba => Soldaki, sağdakinden kalıtım alsın bu şekilde çalışma neticesinde artık compiler sevitesinde Araba sınıfındaki bütün erişime açık olan,kalıtıma müsait olan memberlar Opel'e aktarılmış olacaktır.
* {
* }
* Kodun içinde selena gibi gelmez bu yapmış olduğun işlem neticesinde bir nesne oluşturduğunda ilgili aktarılan/miras alınan datalara/memberlara erişebileceksin Yani bunların direkt fiziksel olarak gözükmesini bekleme bu compiler seviyesinde yani arkada dijital seviyede oluyor.
* Erişim belirliyecileri bir yandan da kalıtımsal durumlarda memberların davranışlarını sergilememizi sağlar
* Kalıtım operasyonel olarak gerçekleştirildikten sonra compiler seviyesinde member aktarımı sağlanır!

Muhasebeci muhasebeci = new Muhasebeci();

Mudur mudur = new Mudur();

Yazilimci yazilimci = new Yazilimci();

class Personel

{

public string Adi { get; set; }

public string Soyadi { get; set; }

public bool MedeniHal { get; set; }

}

class Muhasebeci : Personel

{

//public string Adi { get; set; }

//public string Soyadi { get; set; }

//public bool MedeniHal { get; set; }

public bool Musavir { get; set; }

}

class Yazilimci : Personel

{

//public string Adi { get; set; }

//public string Soyadi { get; set; }

//public bool MedeniHal { get; set; }

public string[] KullandigiDiller { get; set; }

}

class Mudur : Personel

{

//public string Adi { get; set; }

//public string Soyadi { get; set; }

//public bool MedeniHal { get; set; }

}

**Nesne Tabanlı Programlama #17 - Inheritance(Kalıtım) Nedir 2. Bölüm**

**Base Class ve Derived Class Nedir?**

* Kalıtımsal ilişkide olan iki sınıf arasında kalıtım veren sınıfa Base/Parent class Kalıtım alan class'a ise Derived/Child class denir.
* Unutma bir sınıfın sade ve sadece tek bir Base class'ı olabilir! direkt türediği sınıftır Base class'ı
* Base class dediğimiz kavram direkt birebir kalıtımı aldığınız sınıftır.
* Lakin atalarındaki tüm sınıflar Base Class'ı değildir!
* Bir class'ın birden fazla Derived Class'ı olabilir. yani benim bir tane babam vardır ama babamın birden fazla oğlu/kızı olabilir.
* Bir class hem Derived class hem de Base class olabilir
* Bir derived class'ın ataları olabilir fakat onlar derived class'ın base class'ı değildir.Ata yadigarı sınıflar olacaktır.

**Kalıtımın Altın Kuralı!**

* Bir class'ın sade ve sadece bir Base Class'ı olur dedik.
* C# programlama dilinde bir class'ın sade ve sadece tek bir class'tan türetilmesine izin verilmektedir! Aynı anda birden fazla class'tan türeme işlemi gerçekleştirilemez!

**Kalıtımda Nesne Üretim Sırası**

* Bir sınıftan nesne üretimi yapılırken kalıtım aldığı üst sınıflar varsa eğer önce o sınıflardan SIRAYLA nesne üretilir.
* Yazılım sen nesne üretmesende compiler seviyesinde ilgili kalıtım veren sınıftan bir nesne üretilecektir.Öncelikle katlıtım veren daha sınra kalıtım alandan nesne üretilecektir.
* Kalıtımsal ilişkinin olduğu durumda nesne üretiyorsan eğer Ram'in Heap bölgesinde kaç tane hiyerarşik ata varsa o kadar nesne üretilmiş olur
* Bu da arka planda bir sınıftan aktarılmış olan herhangi bir member'ı sen o sınıfta üst sınıflardan gelen bir member'ı kullanmak istiyorsan özünde arka planda oluşturulan nesne üzerinde tetikleme yapıyorsun.
* Yani buradan anlaşılıyor ki bir sınıftan nesne üretilirken siz 1 tane nesne ürettiğinizi düşünsenizde kalıtımsal açıdan birden fazla nesne üretimi gerçekleşebilmektedir.

new D();

class A

{

public A()

{

Console.WriteLine($"{nameof(A)} nesnesi oluşturulmuştur.");

}

}

class B : A

{

public B()

{

Console.WriteLine($"{nameof(B)} nesnesi oluşturulmuştur.");

}

}

class C : B

{

public C()

{

Console.WriteLine($"{nameof(C)} nesnesi oluşturulmuştur.");

}

}

class D : C

{

public D()

{

Console.WriteLine($"{nameof(D)} nesnesi oluşturulmuştur.");

}

}

**Bir Sınıftan Base Class Constructor'ına Ulaşım**

* Madem ki herhangi bir sınıftan nesne üretimi gerçekleştirilirken öncelikle base class'ından nesne üretiliyor, bu demektir ki önce base class'ın constructor'ı tetikleniyor.
* Haliyle bizler nesne üretimi esnasında base class'ta üretilecek olan nesnenin istediğimiz constructor'larını tetikleyebilmeli yahut varsa parametre bu değerleri verebilmeliyiz.
* İşte bunun için base Keyword'ünü kullanmaktayız.
* Eğer ki base class'ın constructor'ı sadece parametre alan constructor ise derived class'larda o constructor'a bir değer GÖNDERMEK ZORUNDAYIZ bunuda base keywordüyle sağlayabiliriz...
* Eğer ki base class'ta boş parametreli bir constructor varsa derived class'ta base ile bir bildirimde bulunmak zorunda değiliz... Çünkü varsayılan olarak kalıtımsal durumda base class'taki boş parametreli constructor tetiklenir
* Bir class'ın constructor'ının yanında : base(...) kullanırsak eğer o class'ın tüm constructorlarını bize getirecektir. Haliyle ilgili sınıftan bir nesne üretilirken base class'tan nesne üretimi esnasında hangi constructor'ın tetikleneceğini bu şekilde belirleyebiliriz...

new MyClass2();

class MyClass

{

public MyClass(int a)

{

}

public MyClass(string a)

{

}

public MyClass(int a, string b)

{

}

public MyClass()

{

}

}

class MyClass2 : MyClass

{

public MyClass2() : base(5, "deasafasf")

{

}

public MyClass2(int a) : base(a)

{

}

}

**base Keyword vs this Keyword**

* this bir sınıftaki constructor'lar arasında geçiş yapmamızı sağlar
* base bir sınıfın base class'ının constructor'larından hangisinin tetikleneceğini belirlememizi ve varsa parametrelerinin değerlerinin derived class'tan verilmesini sağlar.
* thisi kullanıyorsam buradaki constructorlar base'i kullanıyorsam bir üstteki/base class'taki constructorlar
* Ayrıca nasıl ki this keywordü ilgili sınıfta o ank, nesnenin memberlarına erişebilmemizi sağlıyor aynı şekilde base'de base class'taki memberlara erişebilmemizi sağlamaktadır.
* Base class'ta erişilebilir olmayan member'lar base keywordüyle erişilemez!
* private olan bir member base class'tan derived class'a aktarılmıyacağından dolayı base. dediğinde erişilmeyecektir.

class E

{

int a;

public int b;

public int MyProperty { get; set; }

}

class F : E

{

int c;

public void X()

{

MyProperty = 123;

}

}

class BuyukBaba

{

//Büyükbabanın derived classı Baba'dır ama dolaylı yoldan da Ogul class'ı büyükbabanın derived class'ı olabilir. Dolayısıyla aynı mantık bir sınıfın derived class'ları bizzat kendisinden türeyenlerdir

}

class Baba : BuyukBaba

{

}

class Anne : BuyukBaba

{

}

class Ogul : Baba

{

}

**Nesne Tabanlı Programlama #18 - Inheritance(Kalıtım) Nedir 3. Bölüm**

**Bir Nesnedeki ToString, Equals, GethashCode, GetType Metotları Nereden Gelmektedir?**

MyClass myClass = new MyClass();

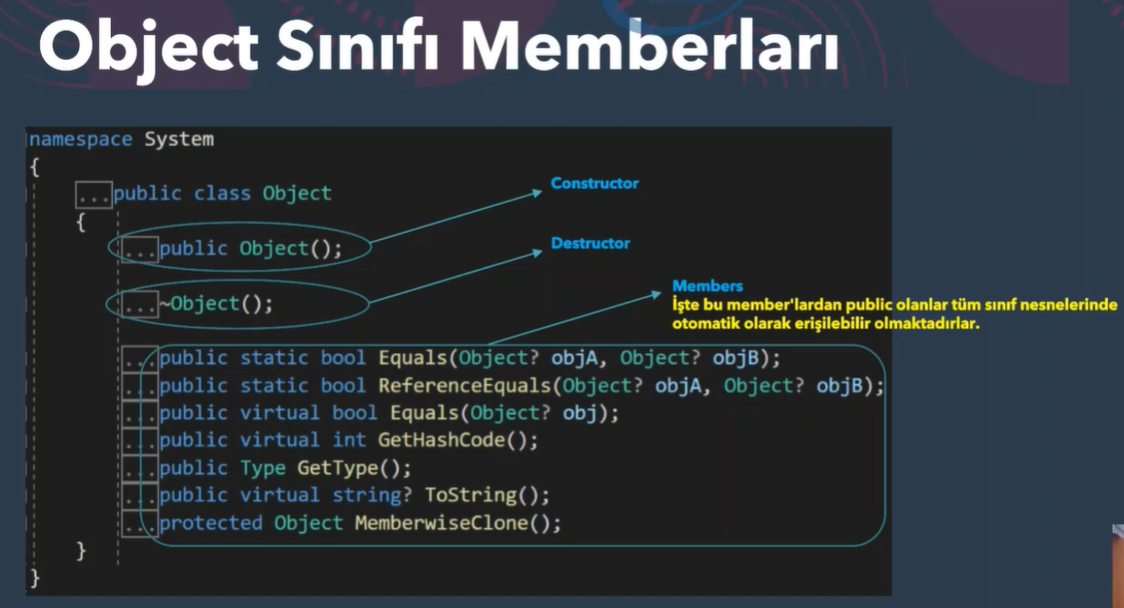
class MyClass

{

}

**Nesnelerin Atası/Ademi Object Türü**

* C# programlama dilinde tüm sınıflar Object sınıfından türetilir
* Bir sınıf oluşturduğunda bir sınıftan türese de türemese de bu sınıf object sınıfından türeyecektir. Yani compiler seviyesinde otomatik olarak default olarak object sınıfından türetilecektir.
* object her şeyi kapsayan bütün değerleri kapsayabilen bir özellik olmasının altında yatan sebep tüm değerlerin Object'ten türemesidir.
* Eğer ki tanımlanan sınıf herahngi bir kalıtım almıyorsa default olarak Object sınıfından türetilecektir. Yok eğer herhangi bir sınıftan kalıtım alıyorsa bir sınıfın aynı anda birden fazla sınıftan kalıtım alamama prensibinden yola çıkarak bir yandan da Object sınıfından türemeyecek sadece kalıtım aldığı sınıftan türeyecektir. ama yine dolaulı yoldan Object sınıfından türemiş olacaktır Object sınıfı atası olmuş olacaktır.
* En nihayetinde bir sınıf öyle ya da böyle Object sınıfından kalıtım alacaktır.



**İsim Saklama(Name Hiding) Sorunsalı**

* Kalıtım durumlarında atalardaki herhangi bir member ile aynı isimde member'a sahip olan nesneler olabilmektedir.
* Bu şekilde aynı isme sahip olan memberlar için base'deki member saklanır.

D d = new D();

class A

{

public void X()

{

}

}

class B : A

{

}

class C : B

{

}

class D : C

{

public new void X()

{

}

}

**Record'lar da Kalıtım**

* Recorlar sade ve sadece Record'lardan kalıtım alabilmektedirler.
* Class'lardan kalıtım alamazlar yahut veremezler!
* Kalıtımın tüm temel kuralları record'lar için geçerlidir.
* Bir record aynı anda birden fazla record'dan kalıtım alamaz!
* Record'larda temelde class oldukları için üretilir üretilmez otomatik olarak Object'ten türerler
* base ve this keyword'leri aynı amaçla kullanılabilmektedir
* Name Hiding söz konusu olabilmektedir.

**Nesne Tabanlı Programlama #19 - Sanal Yapılar virtual - override**

**Sanal Yapılar Virtual & Override**

* Bir nesne üzerinde var olan tüm memberların tamamı derleme zamanında belirgindir.
* Yani derleme aşamasında hangi nesne üzerinden hangi metotların çağrılabileceği bilinmektedir.
* Sanal yapılar ile derleme zamanında kesin bilinen bu bilgi run time(çalışma zamanı)da belirleneblmektedir. Yani ilgili nesnenin hangi metodu kullanacağı bilgisi kararlaştırılır
* Yani bir member'ın bir sınıfın içerisinde olup olmayacağını runtime'da/çalışma zamanında belirlemek istiyorsan bunu biz sanal yapılanmalar sayesinde sağlıyoruz.

**Sanal Yapılar**

* Sanal yapılar bir sınıf içerisinde bildirilmiş olan ve o sınıftan türeyen alt sınıflarda da tekrar bildirilebilen yapılardır.
* Sanal dediğimiz yapı bir sınıfın içinde bildirilmiş olan bir member'dır ve bu sınıftan türeyen başka bir sınıfta tekrardan da bildirilebilir.Biz bunun farkına/hangi sınıfta bildirildiğinin farkına derleme zamanında varamadığımız için runtime'da farkına varırız.
* Bu yapılar metot ya da property'ler olabilir.
* Name hiding base class'ta tanımlanmış olan bir member'ın kendisinden türeyen sınıfların herhangi birinde tekrardan tanımlanmısıdır.Ama sanal yapılanmalar base class'ta tanımlanmış olan sanal yapılanmanın kendisinden türeyen soyundan gelen herahngi bir class'ta bildirilebilmesidir. yani ezilebilmekte/devre dışı bırakılıp yeniden oluşturulabilmektedir.
* Name hiding durumunda var olan bir member aynı isimde kendisinden türeyen farklı bir sınıfta aynı isimde aynı imzada tanımlanır. Amma velakin sen var olan bir member'ı kendisinden türeyen herhangi bir sınıfta ezip içeriğini yeniden oluşturmak/var olan algoritmasını/işlevselliğini/davranışını yeniden şekillendirmek istiyorsan biz buna sanal yapılanma diyoruz
* Name hiding isim çakışmasıdır yani atalarda var olan aynı isimde olan bir member torunlarda herhangi birinde de vardır ikisininde davranışı farklı olabilir bu torunlardaki member atalarda bulunan memberı eziyor/yeniden düzenliyor anlamına gelmez.
* Atalardaki herahngi bir member'ı içeriğinin/algoritmasının/davranışının bilinçli bir şekilde torunlarda değiştirilmesi sanal yapılarla sağlıyoruz bu şu demek atalardaki bu özellik devre dışı
* İşte burada bir sınıfta tasarlanmış sanal yapının işlevinin iptal edilip edilmeme durumuna göre tanımlandığı sınıftan mı yoksa bu sınıfın torunlarından mı çağrılacağının belirlenmesi runtime'da gerçekleşecektir.
* base class'taki atalardaki herhangi bir member sanalsa ve bu sanal yapı atadan mı gelecek yoksa torunların herhngibirinden mi gelecek ordan mı çağrılacak nerden çağrılacağı derleme aşamasında bilinmez.Biz bunun kararını runtime'da veririz.
* Eğer sanallaştırılmış member ezilirse artık fiziksel bir member olmuştur.
* Vee unutma Metot ya da property fark etmez Bir sanal member'ın hangi türe ait olduğunun bu şekilde run time'da belirlenmesine Geç Bağlama(Late Binding) denir.

**Sanal Yapılar Ne İçin Kullanılamaktadır?**

* Kalıtımsal olarak gelen bu niteliği kendimde değiştiriyorsam işte bu değişiklik hakkını bana veren özellik o member'ın/özelliğin/niteliğin sanal olması
* Sanal yapı kendinden türeyen torunlarda ezilmek/yeniden yazılmak zorunda değildir.
* Virtual ile işaretlenmemiş member kesinlikle override edilemez.

**Bir Sınıfta Sanal Yapı Nasıl Oluşturulur ve Ezilir? Vırtual Keywordü**

* Bir sınıfta sana yapı oluşturabilmek için ilgili memberın (metot ya da property ) imzasını virtual keywordü ile işaretlemek yeterlidir.
* Bir classta virual işaretlenerek sanal hale getirilmiş bir member(metot ya da property) bu class tan miras alan torunlarında ezilmek/ yeniden yazılmak isteniyorsa eğer ilgili classta imzası override keywordü işaretlenmiş bir vaziyette tekrardan aynı member oluşturulur.
* Base classta yada atalarda virual ile işaretlenerek sanallaştırılmış herhangi bir member torunlarda illa ki override ile ezilmek zorunda değildir!
* Ama bir torun class base classtaki herhangi bir memberı override edecekse eğer o memberın kesinlikle virtual ile işaretlenmi olması gerekmektedir.

Terlik t = new Terlik();

t.Bilgi();

Kalem k = new Kalem();

k.Bilgi();

class Obje

{

virtual public void Bilgi()

{

Console.WriteLine("Ben bir objeyim...");

}

}

class Terlik : Obje

{

public override void Bilgi()

{

Console.WriteLine("Ben bir terliğim...");

}

}

class Kalem : Obje

{

public override void Bilgi()

{

Console.WriteLine("Ben bir Kalemim...");

}

}

Maymun m = new Maymun();

m.Konus();

Inek i = new Inek();

i.Konus();

class Memeli

{

virtual public void Konus()

{

Console.WriteLine("Ben konuşmuyorum...");

}

}

class Maymun : Memeli

{

public override void Konus()

{

Console.WriteLine("Ben maymunum...");

}

}

class Inek : Memeli

{

public override void Konus()

{

Console.WriteLine("Ben ineğim...");

}

}

Ucgen u = new Ucgen(3, 4);

Console.WriteLine(u.AlanHesapla());

Dortgen d = new Dortgen(3, 4);

Console.WriteLine(d.AlanHesapla());

Dikdortgen dd = new Dikdortgen(3, 4);

Console.WriteLine(dd.AlanHesapla());

class Sekil

{

protected int \_boy; //protected ile işaretlenmiş herhangi bir member sade ve sadece ilgili sınıfta yahut o sınıftan kalıtım almış olan sınıfların içerisinde erişilebilir.Amma velakin nesne üzerinden erişilemez!

protected int \_en;

public Sekil(int boy, int en)

{

\_boy = boy;

\_en = en;

}

virtual public int AlanHesapla()

{

return 0;

}

}

class Ucgen : Sekil

{

public Ucgen(int boy, int en) : base(boy, en)

{

}

public override int AlanHesapla()

{

return \_boy \* \_en / 2;

}

}

class Dortgen : Sekil

{

public Dortgen(int boy, int en) : base(boy, en)

{

}

public override int AlanHesapla()

{

return \_boy \* \_en;

}

}

class Dikdortgen : Sekil

{

public Dikdortgen(int boy, int en) : base(boy, en)

{

}

public override int AlanHesapla()

{

return \_boy \* \_en;

}

}

**Nesne Tabanlı Programlama #20 - Polimorfizm Nedir Derinlemesine İnceleme - 1. Bölüm**

**Çok Biçimlilik(Polimorfizm)**

* Bir nesnenin birden fazla türle/referansla işaretlenebilme mevzusu

**Polimorfizm Nedir?**

* Polimorfizm esasında kalıtım gibi biyolojik bir terimdir.
* Bir olguyu bir nesneyi tarifle/formla tarif edebilmektir.
* İki ya da daha fazla nesnenin aynı tür sınıf tarafından karşılanabilmesidir/referans edilebilmesidir.
* Bir nesnenin birden fazla farklı türdeki referans tarafından işaretlenebilmesi polimorfizm'dir.
* Polimorfizm bir nesnenin kalıtımsal olarak ilişkişi olan diğer nesnelerin referanslarıyla işaretlenebilmesini sağlar
* Polimorfizm OOP tasarımlarında geliştirilen koda daha manevrasal bir şekilde nitelik kazandıran ve yaklaşım sergilememizi sağlayan bir özelliktir.
* Polimprfizm programlamada ki temel prensip olan Sol/Sağ Prensibini aşıp eldeki nesnenin birden fazla referansla işaretlenebilmesini sağlar.
* Polimorfizm dışında elindeki herhangi bir nesneyi sadece o nesnenin referansıyla işaretleyebiliyorsun Amma velakin polimorfizm sayesinde elindeki bir nesneyi o nesnenin türünün dışındaki referanslarla da işaretleyebiliyorsun.İşte bu nesneye farklı türlerdeki referanslarla işaretlenebilme niteliği kazandıran yapılanmaya polimorfizm denir.
* bir nesnenin birden fazla referansla işaretlenmesi o nesnenin birden fazla türün davranışını sergilemsini sağlar.

**Polimorfizm Felsefesi - 1**

* Yani bir olguyu çoklu form olarak tarif edebilmektir.
* Yazılımda polimorfizm'in olabilmesi için kalıtım şarttır
* Bir nesnenin farklı bir nesnenin referansıyla işaretlenebilemsi için o nesneden türüyor olması lazım.

**Polimorfizm Felsefesi - 2**

* Haliyle "Yemek yiyen canlılar yediklerini sindirirler" dediğimizde bir olguya çoklu formları tarif etmiş oluyoruz... İşte burada polimorfizm vardır.

**Polimorfizm Felsefesi - 3**

* Eşşeğe altım semer vursakta eşşek yine eşşşeeeeekkk

**Programlamada Polimorfizm Nerede Kullanılmaktadır?**

* Programlama da polimorfizm esasında taa en temelden beri kullanılmaktadır.
* Örneğin elimizdeki herhangi bir byte türündeki veriyi ister byte istersekte byte'dan büyük olan herhangi bir türde tutmak çok biçimliliktir.
* Ya da object türünün herhangi bir türdeki değeri alabilmesi yahut bir başka deyişle object türüne herhangi bir türdeki veriyiy atayabilmek polimorfizm'dir.
* Evet bir nesnenin başka bir nesne ile işaretlenebilmesi/referans edilebilmesi için kesinlikle arada kalıtımsal bir ilişki olması gerekmektedir.
* Yani bir başka deyişle Nesne tabanlı programlama'da polimorfizm uygulamak istiyorsanız türler arasında kalıtım uygulanmış olmalıdır.
* OOP'de polimorfizm aralarında kalıtımsal ilişki olan sınıflarda uygulanabilir.Aksisi mümkün değildir.

MyClass2 m = new MyClass();

class MyClass : MyClass2

{

}

class MyClass2

{

}

**Polimorfizm Kalıtım İlişkisi**

* Bir nesneyi kendi türünün dışındaki bir tür ile/referansla işaretleyebilmek için kesinlikle ilgili nesne, o türden türemiş olması gerekmektedir.
* Bir sınıftan üretilen bir nesne ancak ve ancak kendisinden olan bir referansla işaretlenebilir ya da atalarından olan bir referansla işaretlenebilir!!!! bu durumda derived class aynı zamanda base class olacağından işaretlenebilir olacaktır.

A1 a = new B1();

B1 b = new B1();

C1 c = new B1();

class A1 : C1

{

}

class B1 : A1

{

}

class C1

{

}

**Polimorfizm Nesne Yönetiminde Neye Yarar?**

* Bir nesnenin birden fazla referansla işaretlenebilmesi o nesnenin birden fazla türün davranışlarını gösterebilmesini sağlar.
* Sen hangi referansla nesneyi işaretlersen sadece O referansın davranışlarını görebilirsin. ya da bu nesnenin referans türündeki davranışlarına ulaşabiliyorsun. Bu referansla nesnenin üzerindeki davranışı elemiş oldun.Çünkü referansın erişebileceğin davranışları sınırlandırarak/filtrelendirerek getirir.
* Polimorfizm bir nesnenin kendi türünün dışında bir veya birden fazla türle işaretlenebilmesidir/referans edilebilmesidir.Ve bunun bize getirmiş olduğu farklı davranışları sergileyebilme niteliğidir.

C c1 = new C();

B b1 = new C();

A a1 = new C();

class A

{

public void X() { }

}

class B : A

{

public void Y() { }

}

class C : B

{

public void Z() { }

}

**Nesne Tabanlı Programlama #21 - Polimorfizm Türleri ve Tür Dönüşüm Operatörleri - 2. Bölüm**

**Polimorfizm Türleri**

**Static Polimorfizm**

* Derleme zamanında sergilenen polimorfizmdir.Hangi fonksiyonun çağrılacağına derleme zamanında karar verir
* C#'ta static polimorfizm deyince aklımıza Metot Overloading gelmelidir.
* Metot overloading aynı isimde birbirinden farklı imzalara sahip olan metotların tanımlanamsıdır.Ya da başka bir deyişle bir isme birden fazlar farklı türde metot yüklemektir.Haliyle burada bir metodun birden fazla formunun olması polimorfizm'ken bunlardan kullanılacak olanın derleme zamanında bilinmesi statik polimorfizm olarak nitelendirilir.

Matematik m = new Matematik();

m.Topla(1, 4, 6, 4);

class Matematik

{

public long Topla(int s1, int s2)

=> s1 + s2;

public long Topla(int s1, int s2, int s3)

=> s1 + s2 + s3;

public long Topla(int s1, int s2, int s3, int s4)

=> s1 + s2 + s3 + s4;

}

**Dinamik Polimorfizm**

* Çalışma zamanında sergilenen polimorfizmdir.Yani hangi fonksiyonun çaışacağına run time'da karar verilir.
* C#'ta dinamik polimorfizm deyince akla Metot Overrride gelmektedir.

Arac a = new Arac();

a.Calistir();

Arac t = new Taksi();

t.Calistir();

class Arac

{

public virtual void Calistir()

=> Console.WriteLine("Araç Çalıştı...");

}

class Taksi : Arac

{

public override void Calistir()

=> Console.WriteLine("Taksi Çalıştı...");

}

* Base class'larda tanımlanmış olan herhangi bir virtual yani sanal yao-pılanma derived class'ta ezilip ezilmediğinin durumu çalışma zamanında bakılıyor.
* Base class'taki sanal bir yapılanmanın Derived class'larda ezilip ezilmediğini/override edilip edilmediğini kararı run time'da veriliyor.
* Metot Override base class'ta virtual olarak işaretlenmiş metotların derived class'ta override edilerek ezilmesi/yeiden yazılması işlemidir.Haliyle burada aynı isimde burdan fazla forma sahip fonksiyonun olamsı polimorfizm'ken bunlardan hangisinin kullanılacağının çalışma zamanında bilinmesi dinamik polimorfizm'dir.

**Polimorfizm Durumlarında Tür Dönüşümleri**

* Polimorfizm OOP'de bir nesnenin kalıtımsal açıdan ataları olan refferanslar tarafından işaretlenebilmesidir..haliyle ilgili nesne bu ataları olan referans türlerine göre dönüştürebilmektedir.
* Dikkat edersen eğer Ğplimorfizm durumlarında kalıtımsal açıdan üst bir referans ile işaretlenebilmiş herhangi bir nesneyi kendi türünden işaretleyebilmek için Cast operatörünü kullanarak object türüne özel olan UnBoxing'e benzer bir hamlede bulunmuş oluyoruz... Buradan anlıyoruz ki object türünde gerçekleştirilen UnBoxing durumu esasında object türü ile gerçekleştirilebilen Polimorfizm'in sonucudur...
* Boxing herhangi bir stack'teki veriyi yani nesne olmayan veriyi nesneye dönüştürebilmek,stackteki herahngi bir türü heap'e alabilmek için yapılan bir uygulamadır.
* Stack'teki tutulan herhangi bir değeri heap'e alabilmek yani primitive bir değeri yani değer türlü bir değeri/türü nesne haline getirebilmek için object mantığını kullanıyoruz.

**Cast Operatörü**

* Eğer ki kalıtımsal ilişki olmayan herhangi bir türe dönüştürülmeye çalışılırsa derleyici hatası verecektir
* Yok eğer kalıtımsal ilişkide olup fiziksel nesnenin hiyerarşik altında olan bir türe dönüştürülmeye çalışılırsa run time hatası verecektir.

A a1 = new C();

//C c = (C)a1;

D d = (D)a1;

//object o = 123;

//int i =(int)o;

class A{}

class B : A{ }

class C : A{ }

class D : C { }

**As İs Operatörü**

* Eğer kalıtımsal ilişkide olup fiziksel nesnenin hiyerarşik altında olan bir türe dönüştürülmeye çalışılırsa run time hatası vermeyecektir geriye null dönecektir.
* as operatörü uygulamayı patlatmaz.
* Cast operatörünün as operatöründen farkı biri dönüşüm sağlanamıyorsa hata fırlatırken(Cast) diğeri null dönmektedir(As).
* İs operatörü kalıtımsal ilişkiye sahip nesnelerin polimorfizm özelliğine nazaran fiziksel olarak hangi türde olduğğunu veren bir opretördür.
* Haliyle çok biçimlilik uyguanmış bir nesnenin ihtiyaç doğrultusunda(uygun olan ) farkı bir türe dönüştürülebilmesi için işi garantiye alabilmek adına önce **İs** konrolü ardından **Cast** yada **as** operasyonu sağlanması kafidir.

A a3 = new C();

C c1 = a3 as C;

Console.WriteLine(a is D); //False

Console.WriteLine(a is C); //True

**Nesne Tabanlı Programlama #22 - Nesneler Arası İlişki Türleri (Association-Aggregation-Composition)**

**Nesneler Arası İlişkiler**

* Nesneler doğrudan ya da dolaylı olarak farklı türlerde birbirleriyle bağ kurabilmektedirler.
* Bu ilişkiler kalıtım, referans yahut soyutlama gibi durumların getirisi olan mantıksal izahatlardır.

**is - a İlişkisi Nedir?**

* is - a ilişkisi tamamıyla kalıtımla alakalıdır. C# programlama dilinde iki sınıf arasında : operatörü ile gerçekleştirilen kalıtım neticesinde ortaya bir is - a ilişkisi çıkmaktadır.
* A is a B => Bu durumda A sınıfı B sınıfından kalıtım almıştır.

**has - a İlişkisi Nedir?**

* Bir sınıfın başka bir sınıfın nesnesine dair sahiplik ifadesinde bulunan ilişkidir. Bir yandan kompozisyon/composition ilişkisi de denmektedir.
* Bir sınıf farklı bir sınıfın referansını kendi içinde barındırıyorsa biz buna has a ilişkisi diyoruz

**can - do İlişkisi Nedir?**

* Interface yapılanamasının getirisi olan bir ilişki türüdür.
* Interface bir sınıfın imzasıdır.Yani bir sınıfın içerisinde olacak olan tüm member'ların şablonunu/arayüznü oluşturduğumuz bir kontrattır.Herhangi bir interface'i uygulayan class o interface içerisinde tanımlanmış member imzalarını kendisinde oluşturmak zorundadır.Aksi taktirde compiler hata verecektir.
* Interfaceler içlerindeki memberların imzalarını class'ta uygulattırdığından dolayı o interfaceler ile ilgili nesnelerin yapabilecekleri kabileyetleri göstermektedir.
* can - do ilişkisi bir nesnenin davranışlarını/kabiliyetlerini belirtmektir. Bu davranış/kabiliyetlerin interface içerisinde tanımlanmaktadır.

**Association Nedir?**

* Sınıflar arasındaki bağlantığının zayıf biçimine verilen addır.
* Bu bağlantı oldukça gevşektir. Yani sınıflar kendi aralarında ilişkilidir lakin birbirinden bağımsızdırlar.
* Parça - bütün ilişkisi oktur
* Örneğin bir otobüsteki yolcular ile otobüs arasındaki ilişki association'dır Nihayetinde hepsi aynı yöne gitmektedir. Lakin bir yolcu indiğinde bu durum otobüsün ve diğer yolcuların durumunu değiştirmez!
* Bir araya gelmiş birçok nesne var serseri bir düzen oluşturmuşlar ama aradan elenenler olduğu zaman bu bütünde herahngi bir aksaklık/eksiklik ortaya çıkmıyor. İşte bu tarz durumlara biz association diyoruz.

**Aggregation ve Composition Nedir?**

* Nesneleri birleştirip daha büyük bir nesne yapmaya verilen isimlerdir.
* Her ikiside birleştirilmiş nesnelerden bütünsel nesneler yapma durumlarını ifade eder
* Her ikisinde de parça - bütün ilişkisi söz konusudur.
* Her ikisinde de sahiplik(has a) ilişkisi vardır.

