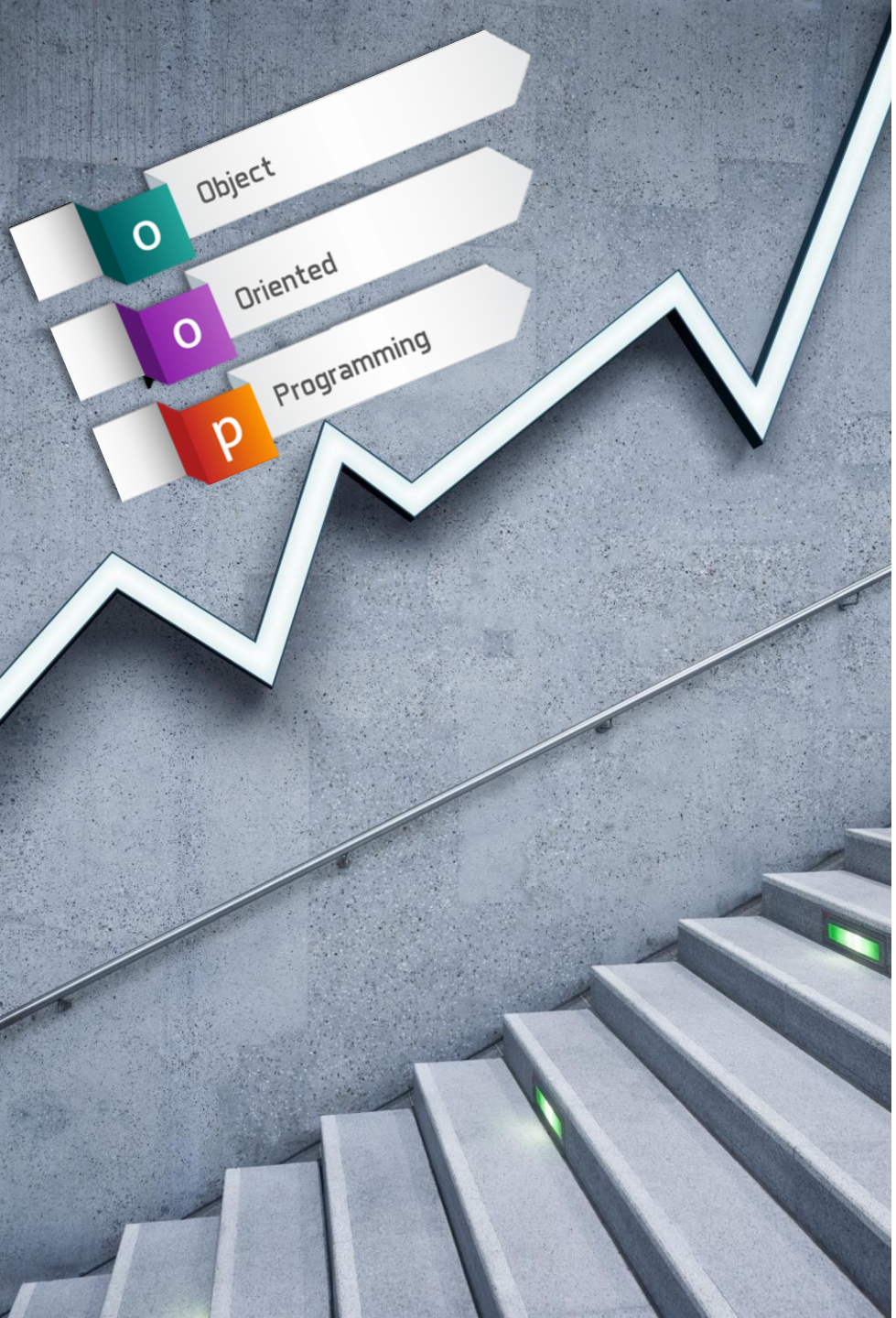
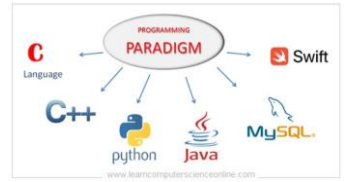
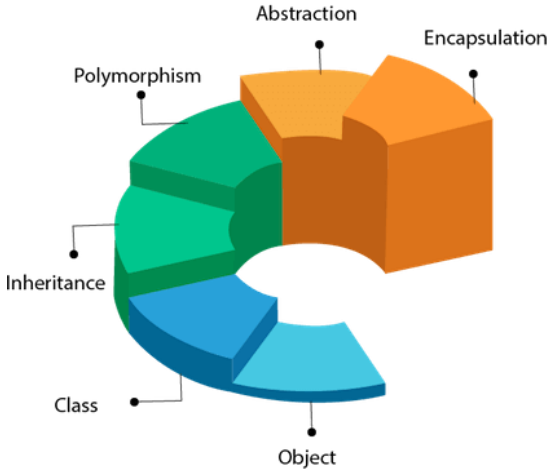
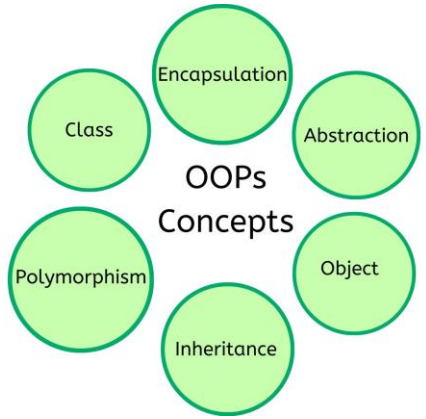
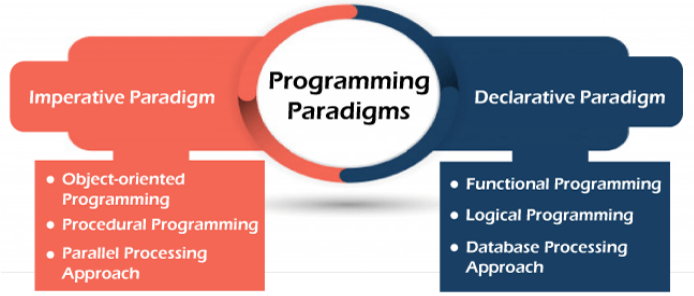


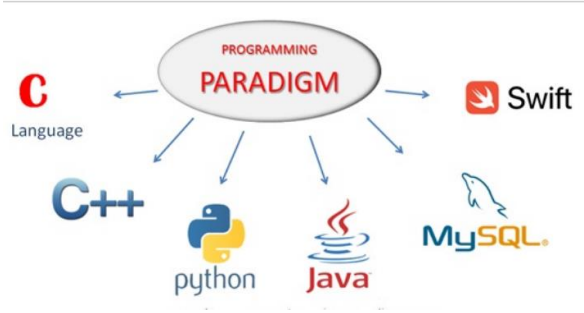
C#
Object Oriented Programming -OOP











ADEM AKKUŞ | Bilgisayar Mühendisi | Uzman Bilişim Teknolojileri Öğretmeni | Eğitmen

Object Oriented Programming

Modern programlama dillerinin çoğu nesne yönelimli programlamayı (NYP) destekler.İlk OOP dil Simula'dır. NYP, yazılım geliştirme sürecini kısaltmakta ve sistematik hale getirmektedir. C# %100 NYP dilidir.

Nesnelerin kaynak koddaki karşılığı sınıflar(class)'dır. Bu bölümde NYP yaklaşımında kullanılan veriyapısı modeli olan sınıfları yakından inceleyeceğiz.

Object Oriented Programming

- Daha esnek kod yazmak.
- Sonradan kullanılabilir kütüphaneler oluşturarak kod tekrarını engellemek.
- Elimizdeki mevcut kütüphanelerden yeni kütüphaneler oluşturmak.

class (Sınıf)

C# ve NYP en önemli veri yapısıdır. Sınıf programcıya bir veri modeli sunar. Sınıflardan nesneler türetilir. Aslında bu işlemi şimdiye defalarca kullandık.

Ancak sınıfların hakkında detaylı bilgiye sahip değildik. Örneğin dizilerin System.Array sınıfından bir nesne olduğunu söylemiştik. Array sınıfının Sort() metodu olduğundan bahsetmiştik. Şimdiye kadar hazır sınıfları ve onların metotlarını, özelliklerini kullanmıştık. Şimdi ise kendi sınıflarımızı oluşturup onları kullanmayı öğreneceğiz.

class (Sınıf)

Class (sınıf) programcının kendi veri türüdür. Sınıflar referans türlerdir. Fakat sınıfın üyeleri değer tipli (int, char, bool vb) olabilir.

```
public  class Ev
{
    string turu;
    int odaSayisi;
    bool kiralikMi;
    double metreKaresi;
}
```

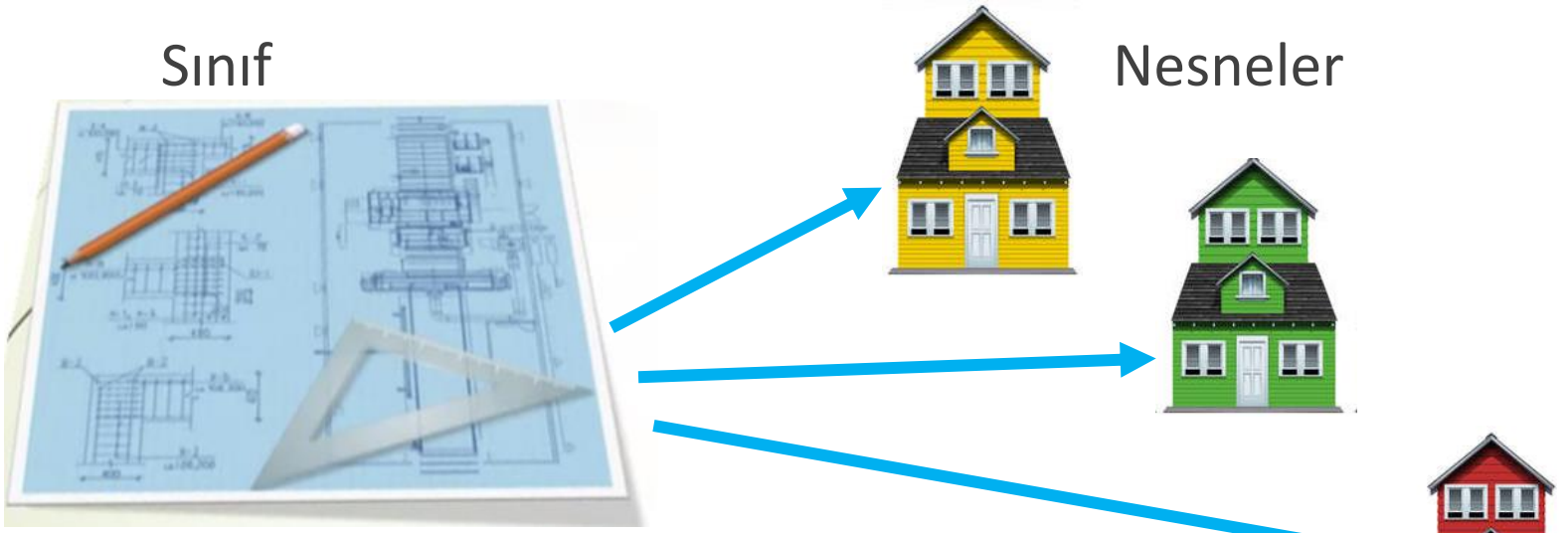
Nesne (object) ise sınıfa değer atadığımız varlıklardır.

Ev ev=new Ev(); //ev nesnesi

Sınıf ile veri yapısının taslağı oluşturulur. Sınıfları kullanarak ortaya bir ürün çıkardığımızda nesneyi oluşturmuş oluruz.

class (Sınıf)

Örnek bir bina projesi sınıf (soyut yapı) , bu projeden yapılan binalar ise nesne (somut yapı)'dir. Binaların hepsinin renk özelliği vardır fakat renkleri farklı olabilir. Bir projeden birden fazla bina oluşturulabilir.



Sınıftan Nesne Tanımlama

Nesne tanımlaması , new anahtar sözcüğü kullanılarak yapılır. new anahtar sözcüğünden sonra kurucu metot (constructor) ismi yazılır.

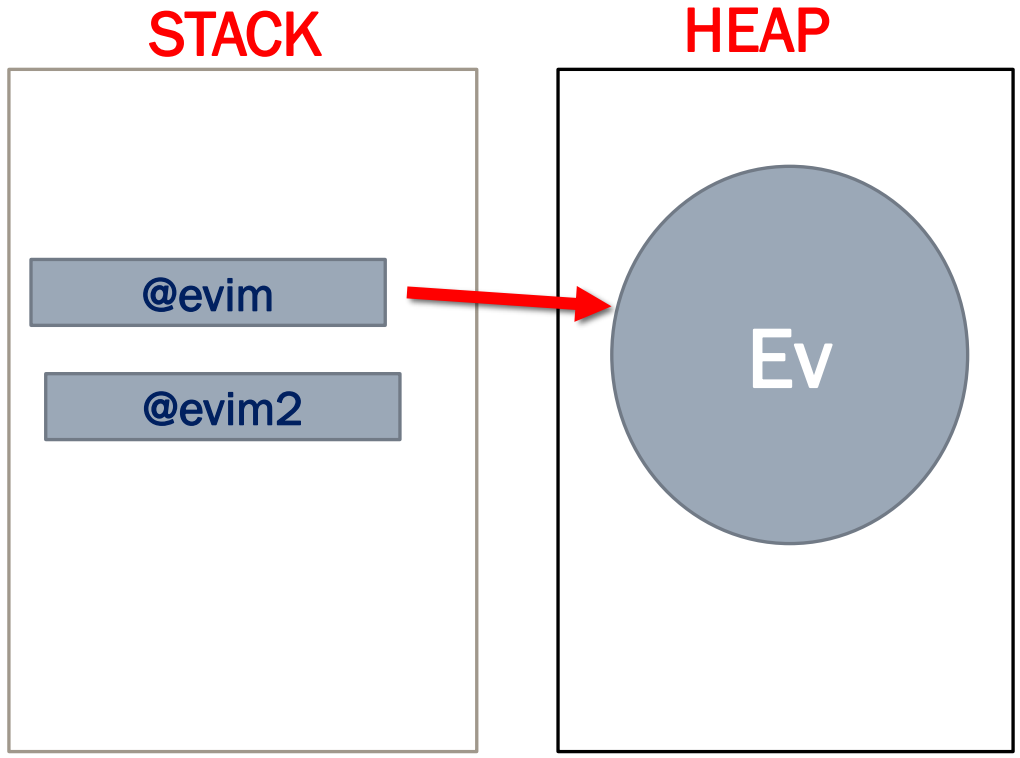
```
Ev evim = new Ev();
```

- `Ev` — Veri Tipi
- `evim` — Nesne (Object)
- `Ev()` — Kurucu Metot

Şu şekilde de tanımlama yapılabilir.

```
Ev evim2;
```

- `Ev` — Veri Tipi
- `evim2` — Nesne (Object)



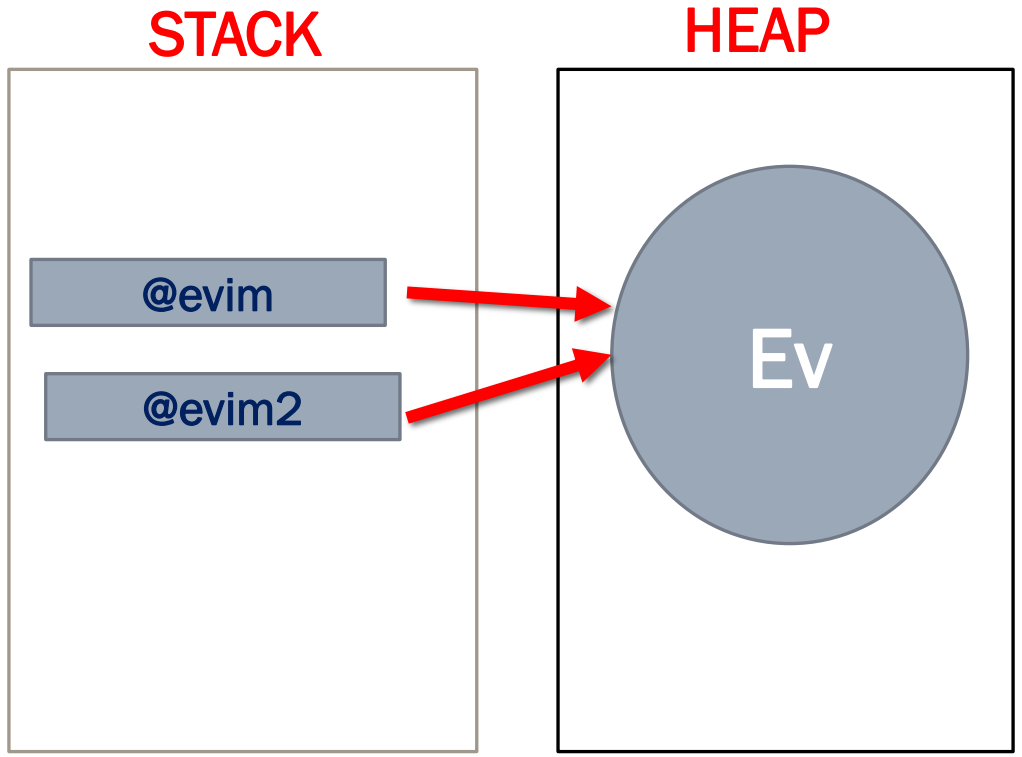
Sınıftan Nesne Tanımlama

Eğer bir nesne kurucu metotu ile oluşturulmazsa

Hiç bire yere referans etmez. Yani null

Referans eder.

```
Ev   evim2;
evim2=evim;
```



class (Sınıf)

Sınıf bildirimi erişim belirleyici, **class** anahtar sözcüğü ile başlar. **class** anahtar sözcüğünden sonra sınıf ismi yazılır. Sınıfın isminden sonra **küme** (süslü, kırlangıç **{ }**) parantezler arasına sınıfın üye elemanları yerleştirilir.

[erişim belirleyici] **class** sınıfIsmi

{

// sınıf üyeleri

}

Not: erişim belireyici yazmak zorunda değiliz. Varsayılan erişim belirleyici neydiii ?

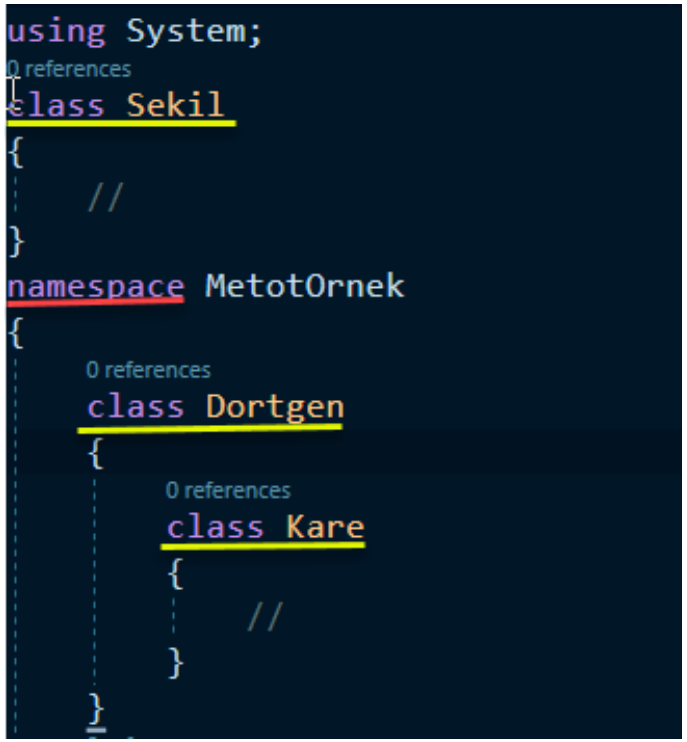
class Nereye Yazılabilir?

Sınıf tanımlaması;

- namespace içerisinde

-namespace dışında

-class içerisinde (nested class)



Not: Aynı kapsam içerisinde aynı isimde iki class tanımlanamaz.

Not: erişim belireyici yazmak zorunda değiliz. Varsayılan erişim belirleyici neydiii ?

Sınıf Üyeleri (Class Members)

Sınıflar ve yapılar, kendi verilerini ve davranışlarını temsil eden üyelere sahiptir. Bir sınıfın üyeleri, devralma hiyerarşisindeki tüm sınıflarda bildirilen tüm üyeler (oluşturucular ve sonlandırıcılar hariç) ile birlikte sınıfında bildirilen tüm üyeleri içerir. Temel sınıflardaki özel üyeler devralınır ancak türetilmiş sınıflardan erişilebilir değildir.

Üye	Açıklama
Alanlar	Alanlar, sınıf kapsamında bildirilen değişkenlerdir. Bir alan, yerleşik bir sayısal tür veya başka bir sınıfın bir örneği olabilir. Örneğin bir takvim sınıfı, geçerli tarihi içeren bir alana sahip olabilir.
Sabitler	Sabitler, değeri derleme zamanında ayarlanan ve değiştirilemeyen alanlardır.
Özellikler	Özellikler, bir sınıftaki alanlar gibi erişilen o sınıftaki öğelerdir. Bir özellik, nesne bilgisi olmadan değiştirilmesini engellemek üzere bir sınıf alanı için koruma sağlayabilir.
Yöntemler (Metotlar)	Yöntemler, bir sınıfın gerçekleştirebildiği eylemleri tanımlar. Yöntemler girdi verilerini sağlayan parametreleri alabilir ve parametreler ile çıktı verilerini döndürebilir. Ayrıca yöntemler parametre kullanmadan bir değeri doğrudan döndürebilir.

Sınıf Üyeleri (Class Members)

Aşağıdaki tablo, bir sınıfın veya yapının içerebileceği üye türlerini listeler:

Üye	Açıklama
Alanlar	Alanlar, sınıf kapsamında bildirilen değişkenlerdir. Bir alan, yerleşik bir sayısal tür veya başka bir sınıfın bir örneği olabilir. Örneğin bir takvim sınıfı, geçerli tarihi içeren bir alana sahip olabilir.
Sabitler	Sabitler, değeri derleme zamanında ayarlanan ve değiştirilemeyen alanlardır.
Özellikler	Özellikler, bir sınıftaki alanlar gibi erişilen o sınıftaki öğelerdir. Bir özellik, nesne bilgisi olmadan değiştirilmesini engellemek üzere bir sınıf alanı için koruma sağlayabilir.
Yöntemler	Yöntemler, bir sınıfın gerçekleştirebildiği eylemleri tanımlar. Yöntemler girdi verilerini sağlayan parametreleri alabilir ve parametreler ile çıktı verilerini döndürebilir. Ayrıca yöntemler parametre kullanmadan bir değeri doğrudan döndürebilir.
Ekinlikler	Olaylar, diğer nesnelere düğme tıklamaları veya bir yöntemin başarıyla tamamlanması gibi örnekleri sağlar. Olaylar, temsilciler kullanılarak tanımlanır ve tetiklenir.
İşleçler	Aşırı yüklenmiş işleçler tür üyeleri olarak kabul edilir. Bir işleci aşırı yüklerken, bunu bir türdeki genel statik yöntem olarak tanımlarsınız. Daha fazla bilgi için bkz . İşleç aşırı yüklemesi.
Dizin Oluşturucular	Dizinleyiciler, bir nesnenin dizilere benzer şekilde dizinlenmesini sağlar.
Oluşturucular	Yapıcılar, nesne ilk oluşturulduğunda çağırılan yöntemlerdir. Bunlar genellikle bir nesneye ait verileri başlatmak için kullanılır.
Sonlandırıcılar	Sonlandırıcılar C# dilinde çok nadir kullanılır. Bunlar, nesne bellekten kaldırılmak üzereyken çalışma zamanı yürütme alt yapısı tarafından çağrılan yöntemlerdir. Bu yöntemler, genellikle yayınlanması gereken tüm kaynakların uygun şekilde işlendiğinden emin olmak için kullanılır.
İç İçe Türler	İç içe olan türler, başka bir türde bildirilen türlerdir. İç içe türler genellikle yalnızca bunları içeren türler tarafından kullanılan nesneleri tanımlamak için kullanılır.

Member	Description
Fields	Fields are variables declared at class scope. A field may be a built-in numeric type or an instance of another class. For example, a calendar class may have a field that contains the current date.
Constants	Constants are fields whose value is set at compile time and cannot be changed.
Properties	Properties are methods on a class that are accessed as if they were fields on that class. A property can provide protection for a class field to keep it from being changed without the knowledge of the object.
Methods	Methods define the actions that a class can perform. Methods can take parameters that provide input data, and can return output data through parameters. Methods can also return a value directly, without using a parameter.
Events	Events provide notifications about occurrences, such as button clicks or the successful completion of a method, to other objects. Events are defined and triggered by using delegates.
Operators	Overloaded operators are considered type members. When you overload an operator, you define it as a public static method in a type. For more information, see Operator overloading.
Indexers	Indexers enable an object to be indexed in a manner similar to arrays.
Constructors	Constructors are methods that are called when the object is first created. They are often used to initialize the data of an object.
Finalizers	Finalizers are used very rarely in C#. They are methods that are called by the runtime execution engine when the object is about to be removed from memory. They are generally used to make sure that any resources which must be released are handled appropriately.
Nested Types	Nested types are types declared within another type. Nested types are often used to describe objects that are used only by the types that contain them.

https://docs.microsoft.com/tr-tr/dotnet/csharp/programming-guide/classes-and-structs/members
https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/members

Sınıf Üyeleri (Class Members)

Aşağıdaki tablo, bir sınıfın veya yapının içerebileceği üye türlerini listeler:

Üye	Açıklama
Alanlar	Alanlar, sınıf kapsamında bildirilen değişkenlerdir. Bir alan, yerleşik bir sayısal tür veya başka bir sınıfın bir örneği olabilir. Örneğin bir takvim sınıfı, geçerli tarihi içeren bir alana sahip olabilir.
Sabitler	Sabitler, değeri derleme zamanında ayarlanan ve değiştirilemeyen alanlardır.
Özellikler	Özellikler, bir sınıftaki alanlar gibi erişilen o sınıftaki öğelerdir. Bir özellik, nesne bilgisi olmadan değiştirilmesini engellemek üzere bir sınıf alanı için koruma sağlayabilir.
Yöntemler	Yöntemler, bir sınıfın gerçekleştirebildiği eylemleri tanımlar. Yöntemler girdi verilerini sağlayan parametreleri alabilir ve parametreler ile çıktı verilerini döndürebilir. Ayrıca yöntemler parametre kullanmadan bir değeri doğrudan döndürebilir.
Ekinlikler	Olaylar, diğer nesnelere düğme tıklamaları veya bir yöntemin başarıyla tamamlanması gibi örnekleri sağlar. Olaylar, temsilciler kullanılarak tanımlanır ve tetiklenir.
İşleçler	Aşırı yüklenmiş işleçler tür üyeleri olarak kabul edilir. Bir işleci aşırı yüklerken, bunu bir türdeki genel statik yöntem olarak tanımlarsınız. Daha fazla bilgi için bkz . İşleç aşırı yüklemesi.
Dizin Oluşturucular	Dizinleyiciler, bir nesnenin dizilere benzer şekilde dizinlenmesini sağlar.
Oluşturucular	Yapıcılar, nesne ilk oluşturulduğunda çağırılan yöntemlerdir. Bunlar genellikle bir nesneye ait verileri başlatmak için kullanılır.
Sonlandırıcılar	Sonlandırıcılar C# dilinde çok nadir kullanılır. Bunlar, nesne bellekten kaldırılmak üzereyken çalışma zamanı yürütme alt yapısı tarafından çağrılan yöntemlerdir. Bu yöntemler, genellikle yayınlanması gereken tüm kaynakların uygun şekilde işlendiğinden emin olmak için kullanılır.
İç İçe Türler	İç içe olan türler, başka bir türde bildirilen türlerdir. İç içe türler genellikle yalnızca bunları içeren türler tarafından kullanılan nesneleri tanımlamak için kullanılır.

Member	Description
Fields	Fields are variables declared at class scope. A field may be a built-in numeric type or an instance of another class. For example, a calendar class may have a field that contains the current date.
Constants	Constants are fields whose value is set at compile time and cannot be changed.
Properties	Properties are methods on a class that are accessed as if they were fields on that class. A property can provide protection for a class field to keep it from being changed without the knowledge of the object.
Methods	Methods define the actions that a class can perform. Methods can take parameters that provide input data, and can return output data through parameters. Methods can also return a value directly, without using a parameter.
Events	Events provide notifications about occurrences, such as button clicks or the successful completion of a method, to other objects. Events are defined and triggered by using delegates.
Operators	Overloaded operators are considered type members. When you overload an operator, you define it as a public static method in a type. For more information, see Operator overloading.
Indexers	Indexers enable an object to be indexed in a manner similar to arrays.
Constructors	Constructors are methods that are called when the object is first created. They are often used to initialize the data of an object.
Finalizers	Finalizers are used very rarely in C#. They are methods that are called by the runtime execution engine when the object is about to be removed from memory. They are generally used to make sure that any resources which must be released are handled appropriately.
Nested Types	Nested types are types declared within another type. Nested types are often used to describe objects that are used only by the types that contain them.

https://docs.microsoft.com/tr-tr/dotnet/csharp/programming-guide/classes-and-structs/members
https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/members

Sınıfa Metot Ekleme

```
class Dortgen
{
    public int en;
    public int boy;
    /// <summary>
    /// Alan hesaplaayn metot
    /// </summary>
    /// <returns>int alan degeri</returns>
    1 reference
    public int Alan()
    {
        int alan = en * boy;
        return alan;
    }
    /// <summary>
    /// Bilgileri ekrana yazdırır.
    /// </summary>
    1 reference
    public void Yazdir()
    {
        Console.WriteLine("---------------");
        Console.WriteLine("En:{0,5}",en);
        Console.WriteLine("Boy:{0,5}",boy);
        Console.WriteLine("Alan:{0,5}",Alan());
    }
```

```
class Program
{
    0 references
    static void Main(string[] args)
    {
        Dortgen dortgen = new Dortgen();
        dortgen.en = 10;
        dortgen.boy = 20;
        //dortgen.Alan();
        dortgen.Yazdir();
    }
}
```

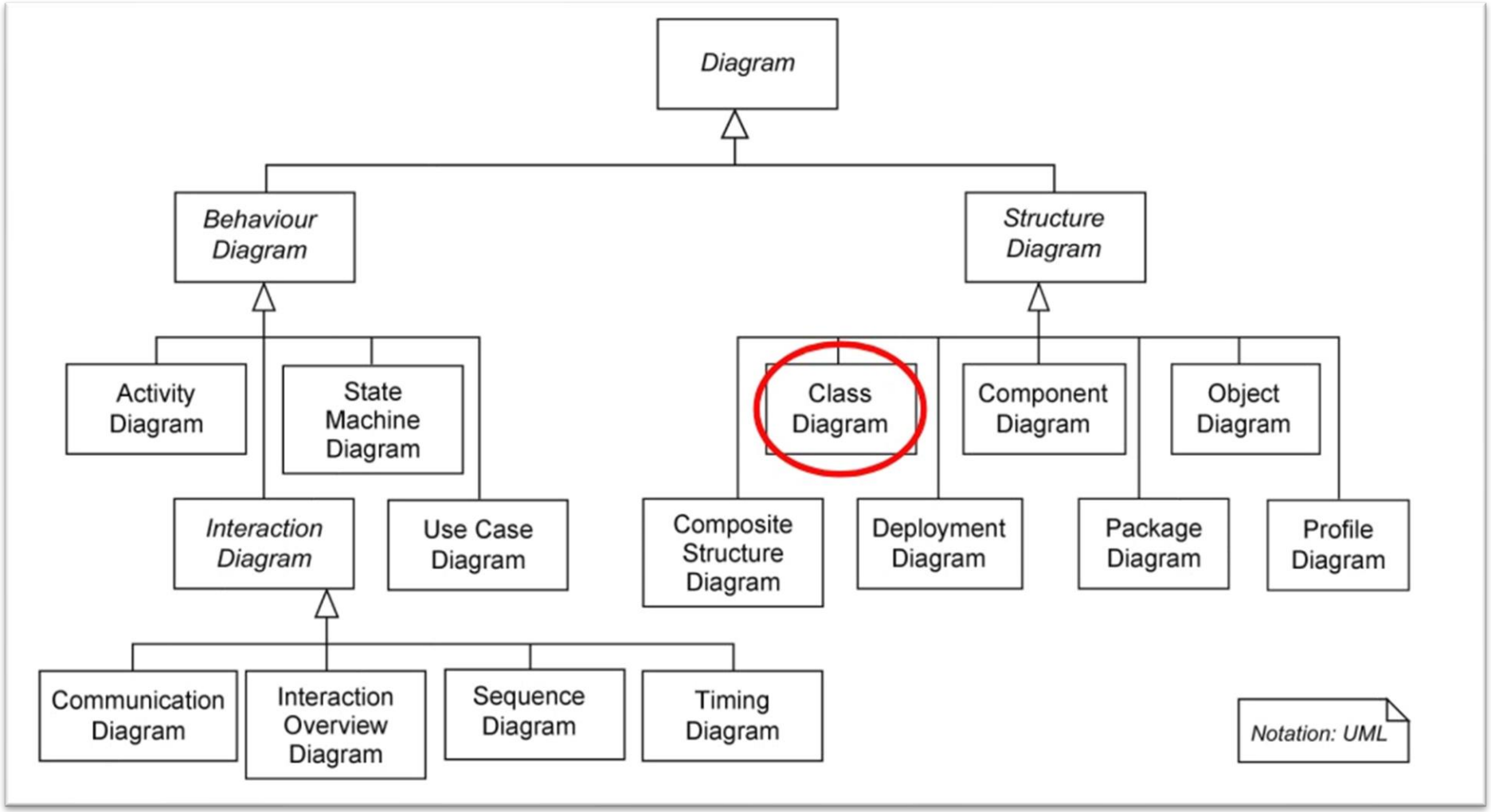
Microsoft Visual Studio Debug Console

```
---------------
En:   10
Boy:   20
Alan:  200
```

UML (Unified Modeling Language) Diagrams

UML, bir sistemin tasarımını görselleştirmek için yazılım mühendisliği alanında genel amaçlı modelleme dilidir. Yazılı bir dil değildir. Farklı amaçlar için kategorilere ayrılmış olsa da, genel itibariyle modelleme için kullanılır. 1995 yılında, yazılımlarda bir standart yaklaşım oluşturmak için geliştirilmiştir. Yani UML diyagramları ile önceden modellediğiniz bir yazılım projesini, modele uygun olacak şekilde herhangi bir dil ile geliştirebiliyorsunuz. Bu da yazılım mühendisleri arasında ortak bir dil oluşturuyor. İlk çıktığı zamandan beri sürekli geliştirme göstererek, birçok farklı dala ayrılmıştır. Aşağıda UML diyagramlarının kategorilerini görebiliyoruz. Biz sadece Class Diagram inceleyeceğiz.

UML (Unified Modeling Language) Diagrams

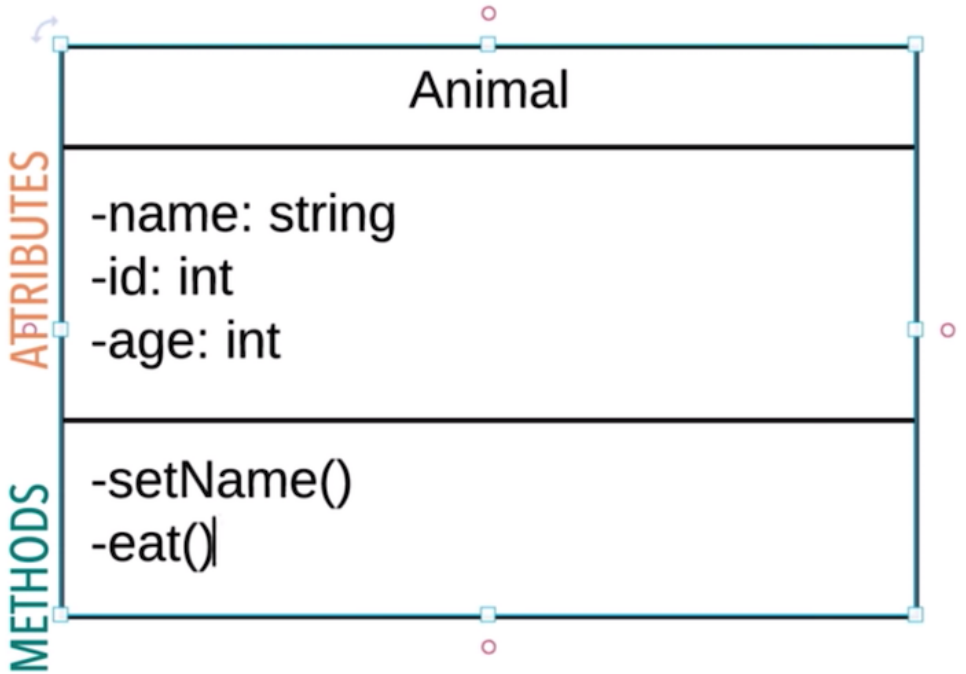


UML Class Diagram

Class diyagramları, OOP temel alınarak tasarlanmıştır. Amaç yazılımımız içindeki sınıflar ve aralarındaki ilişkileri tanımlamaktır. Gelin daha detaya inelim. Aşağıdaki gibi Animal adında bir sınıfımız olduğunu düşünelim.

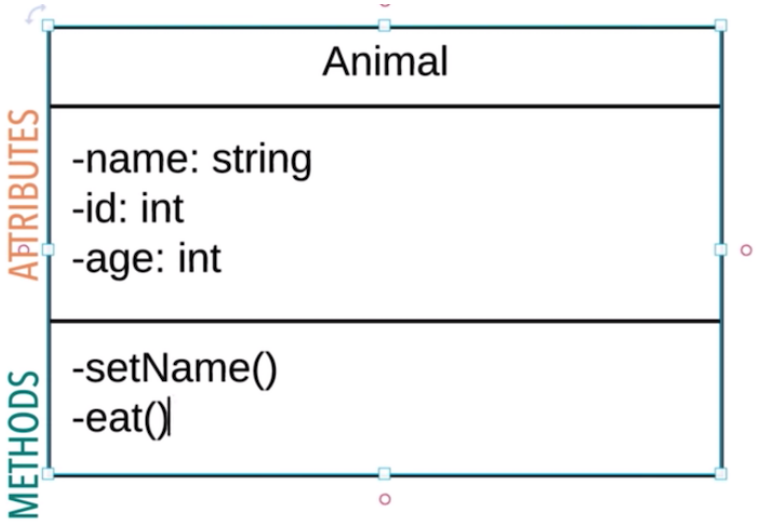
```
class Animal {
      private String name;
      private int id;
      private int age;


      public void setName(String name){
          this.name=name;
      }
      public void eat() {
          System.out.println("Eating");
      }
}
```



UML Class Diagram

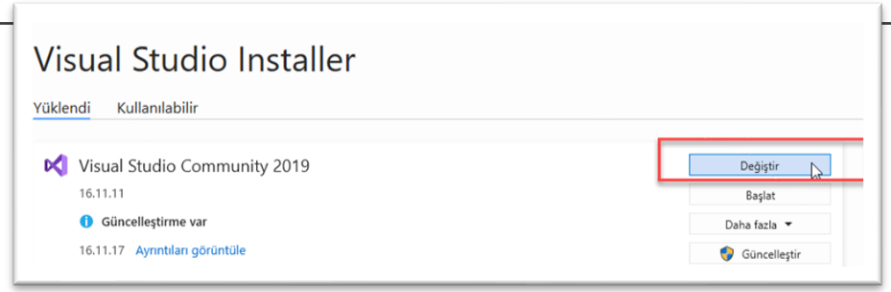
Class diyagramlarında sınıflar bu şekilde ifade ediliyor. Üstte “Attributes” yani sınıfa ait nitelikler(örneğin isim, yaz, id bilgisi), alt kısımda sınıfa ait metodlar bulunur. İfadelerin solunda bulunan “-” işareti ise erişim niteleyicisidir. Sınıf abstract olsaydı *Animal* şeklinde italik yazarak ifade edebilir ya da <<abstract>> şeklinde altına yazabilir. Aynı şekilde bir interface içinse <<interface>> şeklinde belirtiriz.



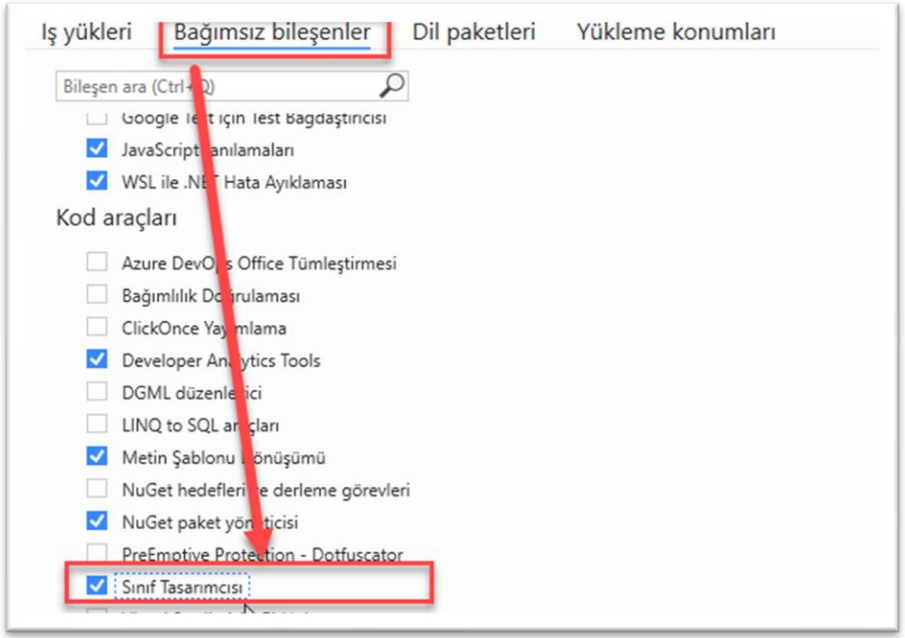
-	private
+	public
#	protected
~	package/default

Visual Studio Class Designer Kurulumu

Visual Studio Installer çalıştırılır. Açılan pencereden «Değiştir» butonuna tıklanır.

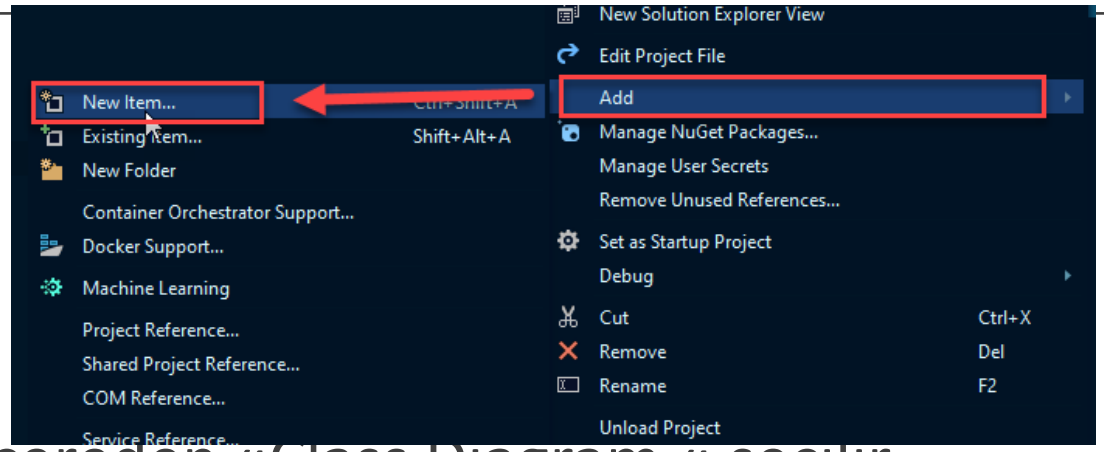


«Bağımsız Bileşenler» sekmesinden «Kod Araçları» grubu altında «Sınıf Tasarımcısı» seçilip yüklenir.

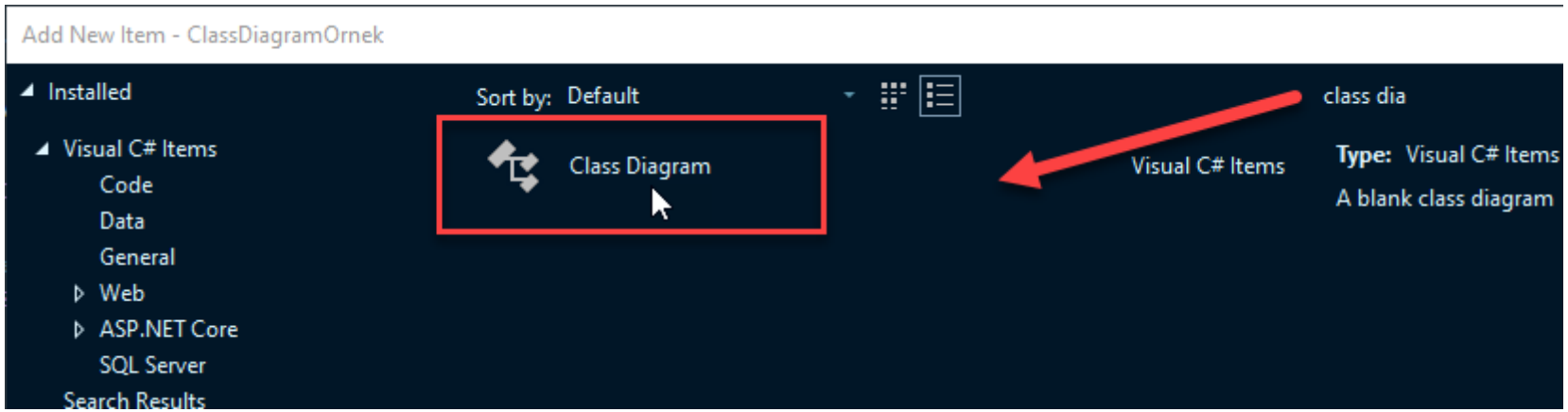


Visual Studio Class Diagram Oluşturma

Proje ismine sağ tıklayıp «Add → New Item» seçilir.

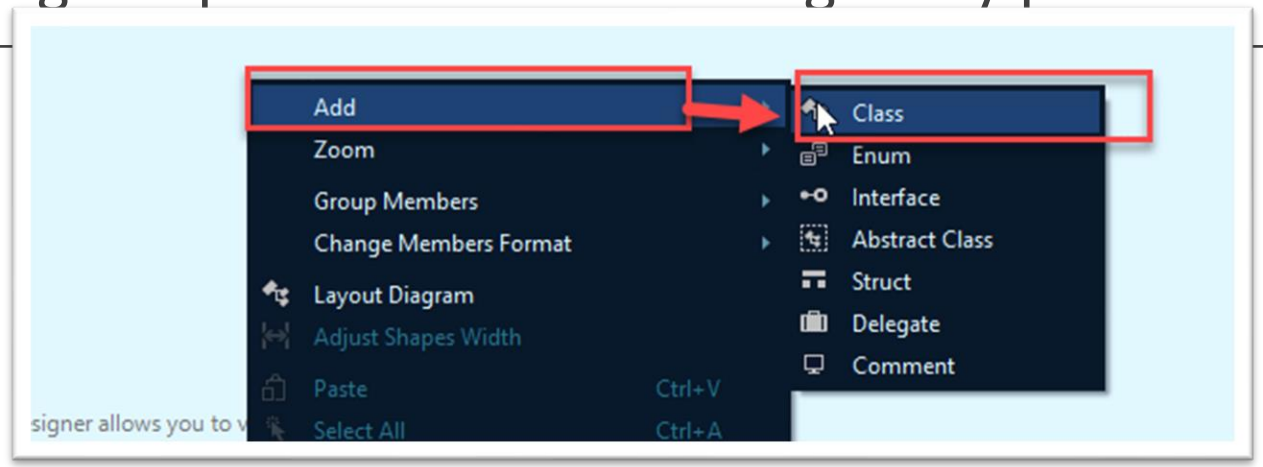


Açılan pencereden «Class Diagram « seçilir.

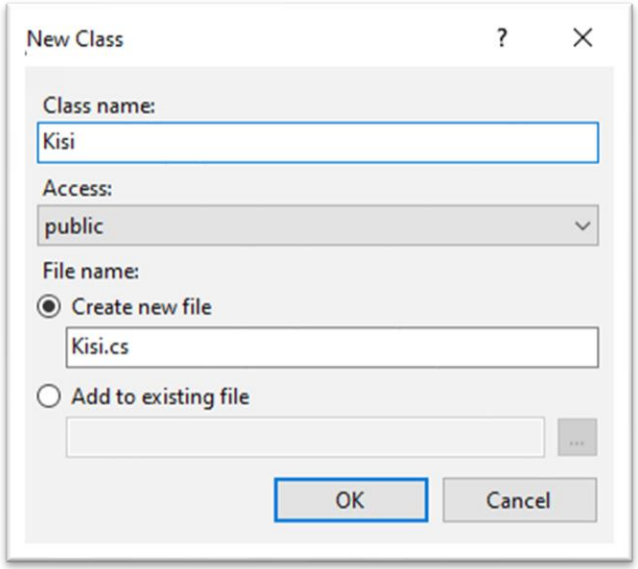


Visual Studio Class Diagram Oluşturma

Açılan class diagram penceresi üzerinde sağ tıklayıp « Add → Class « seçilir.



Açılan pencereden sınıf ismi verilir.



Visual Studio Class Diagram Oluşturma