C# OOP

- https://www.youtube.com/watch?v=NyW4OrFuU8g

Object Oriented Programming'de ana mantık, mevzu bahis konuların classlar ile soyutlanarak hem bilgisayarın hem de insanların anlayabileceği modeller oluşturmaktır.

- **CLASS :** Classlar bizim yapmak isteğimiz işlemleri gruplara ayırmak, o grup üzerinden işlemlerimizi yapmak ve rahatlıkla bu gruba ulaşmak için kullanabilirz.
- bir class'ı kullanabilmek için onun örneğini(referansını) oluşturmamız gerekmektedir.
- bir class oluştururken kelimenin ilk harfi büyük oluşturulur. Ama örneği oluşturulduğunda ilk harfi küçük, sonraki kelimelerin ilk harfi büyük yazılır.
- classlar bir gruplama tekniği olarak metodlar ile kullanılabilir.
- classların bir diğer özelliği property dediğimiz nesneleri(özellikleri) tutmasıdır.
- Property olarak tanımladığımı classlarımızdaki get ve set dediğimiz bloklar, encasulation tekniklerinin en temel versiyonu ile kullanılır.
- bir field üzerinde get veya set ederken yani değeri verirken veya değeri okurken başka bir şey yapmak istersek encapsulation tekniğini kullanmamız gereklidir.

- **INTERFACE** : İsimlendirme standardı **I** ile başlatmaktadır. Interface'lerin en büyük kullanım amacı bir temel operasyon, temel nesne oluşturup bütün nesneleri ondan implemente etmektir.

Interface'ler soyut nesnelerdir. Soyut nesneler tek başına hiç bir anlam ifade etmezler.

Bir interface hiç bir zaman new anahtar sözcüğü ile kullanılamaz(new lenemez). Çünkü tek başına bir anlam ifade etmezler.

Interface ve **abstract** gibi soyut nesneleri **new anahtar sözcüğü ile kullanılamaz.**

- **POLYMORPHISM (ÇOK BİÇİMLİLİK) :** Bir nesneyi farklı amaçlarla implemente edip, o implemantasyonları belli bir kısmına veya tamamına ulaşmak için kullanılır.

Bir class birden fazla interface'i implemente edebilir.

- **INHERITANCE (KALITIM) :** Bir nesneyi bir defa inheritance olarak alabiliriz ama birden fazla implemantasyon yapabiliriz. (Inheritance) önce yazılır

Class'ların tek başına bir anlamı vardır ama **interface**'ler tek başına bir anlam belirtmezler.

- **VIRTUAL METHOD :** Virtual methods (sanal metotlar), base class (temel sınıf) içinde bildirilmiş ve derived class (türemiş sınıf) içinde de tekrar bildirilebilen metotlardır. Böylelikle sanal metotlar kullanılarak nesne yönelimli programlama da çok sık başvurulan çok biçimliliği yani poliformizm (polimorphizm) uygulanmış olur. (virtual - override)
Yani temel sınıfta bir sanal metot bildirildiğinde bu temel sınıftan türeyen sınıflardaki metotlar override edilerek, temel sınıftaki sanal metotu devre dışı bırakabilirler.

Virtual methodlarda temel operasyonlarımız var. İstediğimiz operasyonları override ile ezebiliriz.

- **ABSTRACT CLASS :** Abstract' larda birer classtır, yani interface gibi değil. Interface'ler ile virtual methodların birleşimi olarak düşünebiliriz. Tamamen INHERITANCE(KALITIM) amacı ile kullanılır.Interface'ler ile virtual methodların tam kullanım nedenlerini bir araya getirilmesinden oluşur.

- Abstract method sadece abstract class ta olabilir ve derived class tarafında override edilmek zorunda.
- Virtual method ise normal class ta bulunur ve derived class tarafında override edilmek zorunda değil.

- https://yazilimnedir.com/nesne-yonelimli-programlama-oop-nedir/
- https://yazilimnedir.com/csharp-nesne-yonelimli-programlama-oop-nedir-tum-detaylariyla/

- OOP'nin temel amacı gün geçtikçe ilerleyen ve genişleyen yazılım sektöründeki bir takım problemlere ve tıkanıklıklara çözüm üretmektir.

Object Oriented Programming'de ana mantık, mevzu bahis konuların classlar ile soyutlanarak hem bilgisayarın hem de insanların anlayabileceği modeller oluşturmaktır.

1.Classes & Object (Sınıflar ve Nesneler)
Gerçek hayatta sorun olarak baz alınan modellemeyi ifade eder. Bu sınıflardan aldığımız instance'lar ile de objelerimizi elde ederiz.

```
class OrnekSinif { }
```

Ayrıca class yerine struct terimi ile de böyle bir örnekleme yapabiliriz. Struct ise daha küçük kapsamlı bir öğeyi betimlemek istediğimizde kullanabiliriz.

- **Encapsulation , Properties , Fields (Kapsülleme)**
Encapsulation, bir fielda direk erişimi engelleyerek değimi yerindeyse onu sarmallar ve girdi çıktıları kontrol altına alır.

```
class OrnekSinif
{
  private string exampleField;

  /* okuma ve atama işlemlerinde herhangi bir aksiyon almak
   * istediğimizde 'get' ve 'set' 'leri kullanırız.
   */
  public string exampleProperty { get; set; }

  private string _ornekField;
  public string OrnekProperty
  {
    /* Field'daki data döndürülüyor. */
    get { return _ornekField; }
    /* Field'daki data set ediliyor.. */
    set { _ornekField = value; }
  }
}
```

- **Methods, Overload Methods**
Metodlar, bir sınıfa veya nesneye bağlı olarak çalışıp ilgili eylemleri yerine getirirler.

```
class OrnekSinif
{
  /* erisimBelirteci donusTipi metodAdı(optional Parametreler)
     { } */
  /* parametresiz method */
  public void VoidOrnekMethod()
  {
    Console.WriteLine("Parametresiz Method");
  }

  /* parametreli method
   * (erişimBelirteci belirtilmezse 'private' olarak kabuş edilir.) */
  int IntOrnekMethod()
  {
    Console.WriteLine("Parametreli Method");
    return 0;
  }
}
```

Aynı method isminde ve farklı parametreler ile oluşturduğumuz methodlara ise Overload Method denir.

```
class OrnekSinif
{
  int OverloadMethod()
  {
    return 0;
  }

  int OverloadMethod(int sayi1)
  {
    return sayi1;
  }

  int OverloadMethod(int sayi1, int sayi2)
  {
    return sayi1 + sayi2;
  }
}
```

- **Constructor Methods (Yapılandırıcı/Kurucu Method)**
Class'dan bir instance oluşturulurken ilk olarak Constructor tetiklenir ve çalışır. Bu instance'a ilişkin olarak alınması gereken aksiyonlar da burada yer alır. Ayrıca constructor methodlar overload da edilebilir.

```
class OrnekSinif
{
  public OrnekSinif()
  { Console.WriteLine("Sabit constructor"); }

  public OrnekSinif(string a, string b)
  { Console.WriteLine("Değişken constructor : " + a + " - " + b); }
}
```

- **Nested Classes (İç İçe Sınıflar)**
Bir sınıf içerisine tanımlanmış başka bir sınıfı ifade eder. Basitçe şöyle tanımlanır:

```
class KapsayiciSinif
{
  public class OrnekSinif
  {
    private int _sayi1;
    public int OrnekProperty
    {
      get { return _sayi1; }
      set { _sayi1 = value; }
    }
  }
}
```

- **Access Modifiers (Erişim Belirleyicileri)**
Bir sınıf veya sınıf üyelerinin hangi erişim düzeyinde olacağını belirler.

public : Bulunduğu class ve dışarıdan tam erişim.
private : Dışarıdan hiçbir şekilde erişilemez. Sadece bulunduğu class içerisinde erişilebilir.
protected : Bulunduğu class'dan ve bulunduğu class'da türetilen class'lardan erişilebilir.

- **Static Classes & Members (Statik Sınıflar)**
Statik sözcüğü, sınıflar ve üyeleri için herhangi bir instance işlemine gerek duyulmadan direk erişilebilmesini sağlar.

```
class OrnekSinif
{
  public static string Property1 = "Hello";
  public static string Property2 = "World";

  public static int PlusOne(int number)
  {
    return number + 1;
  }
}
```

kullanımı

```
string metin = OrnekSinif.Property1 + " " + OrnekSinif.Property2;
int sayi = OrnekSinif.PlusOne(5);
Console.WriteLine(metin + " " + sayi);
```

- **Anonymous Types (Anonim Tipler)**
Herhangi bir veri türü veya class belirtmeksizin oluşturulur. Bu işlemi verilen sınıf veya veri türüne göre sistem otomatik olarak yapar. Fakat bizim bir tür belirtmemize gerek kalmaz tanımlama yaparken.<

```
var obj = new { A = "Hello", B = "World" };
Console.WriteLine(obj.A + " " + obj.B);
```

2.Inheritance (Kalıtım, Miras Alma, Devralma)
Bir sınıfın özelliklerinin kalıtım yolu ile başka bir sınıfa aktarılmasıdır.

“ --- Bir class yalnızca bir class’ı miras alabilir. --- ”

```
/* Base Class */
class A
{ }

/* B class'ı A class'ının özelliklerini devralıyor. */
class B : A
{ }

class C : B
{ }
```

Bu noktada iki farklı konu karşımıza çıkıyor. Birincisi “**sealed class**” , ikincisi ise “**abstract class**” kavramlarıdır.

sealed class: sealed olarak tanımlanan bir class, başka bir class’a inheritance(kalıtım, miras alma, devralma) için kullanılamaz. Sadece ilgili bussines’ı veya görevi yerine getirir. Bu sınıfta bulunan herhangi bir method veya member’ın ezilmesini istemeyiz.

abstract class: abstract şekilde tanımladığımız classları base olarak kullanırız. Yani “bir varlığın en temel özelliklerini en ilkel şekilde barındırır” diyebiliriz. Abstract class’lardan instance oluşturulamaz. İçerisine kod yazılabilir. Private olarak tanımlanamazlar.
Temel özellikleri türetilen sınıfta override edip, türetilen class’a uygun şekil almasını sağlayabiliriz.

```
public sealed class D { }
public abstract class E { }
```

- **Overriding (ezme, geçersiz kılma, çiğnemek)**
Base class’dan alınan default özellikleri, türetilen sınıfa göre değiştirmek/geçersiz kılmak istiyorsak bu işlemi uygularız.

C# Modifiers(düzenleyiciler): **virtual**, **override**, **abstract**, **new Modifier**

virtual: Türetilen sınıfta override edilebilmesini sağlar.
override: Türeyen sınıfta override eder.
abstract: Türeyen sınıfta override etmek zorunludur.
new: Türetilen sınıftan devralınan member’ı gizler.

```
class BaseClass
{
  public virtual void OzellikYazdir()
  { Console.WriteLine("Kamil KAPLAN"); }
}

class Example: BaseClass
{
  public override void OzellikYazdir()
  {
    Console.WriteLine("Metodun içindeki değeri ezdik.");
    // base.OzellikYazdir();
  }
}
```

---

3. Interfaces (Arabirimler)
Sınıfların kullanacağı özelliklerin sözleşmesinin yapıldığı ara birimlerdir. Bu arabirimleri referans alan tüm class’lar, ilgili sözleşmedeki tüm member’ları kullanmak zorundadır.

“ --- Bir class birden fazla interface alabilir. --- ”

```
interface ICrud
{
  int sayi1 { get; set; }
  void Ekle();
  void Sil();
  void Duzenle();
  void Listele();
}

class OrnekSinif : ICrud
{
  public int sayi1 { get; set; }

  public void Duzenle()
  { }

  public void Ekle()
  { }

  public void Listele()
  { }

  public void Sil()
  { }
}
```

---

4. Generics
Generic olarak oluşturulacak öğe, hangi türde oluşturulması isteniyorsa o türü parametre olarak alır ve o işlemi gerçekleştirir. Bize şunu sağlar: Tek seferde yazılan bir işlevi farklı obje türleri için de işlevsel hale getirebilme imkanı verir. Yani birçok iş için bir kere efor harcanır. Gereksiz kod tekrarının önüne geçer.

```
/* T tipi alınıyor. */
class ExampleGeneric<T>
{
  public T Insert(T model)
  {
    Console.WriteLine(model);
    return model;
  }
}
```

kullanımı

```
ExampleGeneric<string> example = new ExampleGeneric<string>();
example.Insert("Kamil KAPLAN");
```

- https://medium.com/@berkekurnaz/c-ile-nesne-y%C3%B6nelimli-programlama-26997a747d0c

OOP : 3 ana ilke bulundurmaktadır. Bunlar encapsulation(kapsülleme), polymorphism(çok biçimlilik) ve inheritance(kalıtım)’dır.

- **Class’lar**
Nesne Yönelimli Programlamanın(OOP) temel yapı taşlarıdır. Sınıflar birer veri yapısı olup programcıya bir veri modeli sunar. Bu veri modeli yardımıyla nesneler oluşturulur.

```
class Personel
{
  public string isim;
  public string soyIsim;
  public int yas;
}                          kullanımı
Personel p = new Personel();
p.isim = "Kamil";
p.soyIsim = "KAPLAN";
p.yas = -20;
Console.WriteLine(p.isim + " - " + p.soyIsim + " - " + p.yas);
```

- **Encapsulation(Kapsülleme)**
Sınıf içerisinde yer alan alanların sınıfın dışarından erişiminin kapatılarak kontrol altına alınmasına encapsulation(kapsülleme) diyoruz.

```
class
{
  public string isim;
  private int mYas;

  public int yas
  {
    get
    {
      return mYas;
    }

    set
    {
      if (value < 0 || value > 100)
      {
        value = 0;
      }
      else
      {
        mYas = value;

      }
    }
  }
}  kullanımı

Encapsulation e = new Encapsulation();
e.isim = "Kamil";
e.yas = -23;
Console.WriteLine(e.isim + " - " + e.yas);
```

- **Sınıf İçerisinde Metot Tanımlama**

```
public void BilgileriYazdir()
{
  Console.WriteLine(isim + " " + soyIsim + " "+ yas);
}

public int DogumTarihiniBul(int _yas)
{
  yas = (DateTime.Now).Year - _yas;
  return yas;
}
```

kullanımı

```
p.BilgileriYazdir();
Console.WriteLine(p.DogumTarihiniBul(Math.Abs(p.yas)));
```

- **Yapıcı Metodlar (Constructors)**
Bir nesne dinamik olarak oluşturulduğunda otomatik olarak çalışan metodlardır. Nesnenin elemanlarına ilk değeri vermek için ya da sınıf
nesnesi için gerekli kaynak düzenlemeleri yapma da kullanılır.

```
class MediumPersonel
{
  public string _name;
  public string _lastName;

   public MediumPersonel(string name, string lastName)
   {
     this._name = name;
     this._lastName = lastName;
   }
} kullanımı

MediumPersonel medium = new MediumPersonel("Kamil", "KAPLAN");
```

- **Yapıcı Metotlarda Overloading (Aşırı Yüklenme)**
yapıcı metotlarda overloading(aşırı yüklenme) işlemi nasıl yapılıyor ona bakalım.

```
class MediumPersonel
{
  public string _name;
  public string _lastName;

  public MediumPersonel()
  { }

  public MediumPersonel(string name, string lastName)
  { this._name = name;
    this._lastName = lastName;
  }
}
```

- **Yıkıcı Metotlar (Destructors)**
bir nesne için bellekte ayrılan alanı kaynağa iade etmeden hemen önce çalışır. Bir sınıfın sadece bir tane yıkıcı metodu olabilir ve herhangi bir parametre almaz.

```
~Encapsulation()
{ Console.WriteLine("Burası yıkıcı method"); }
```

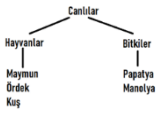
- **Statik Metotlar**
Statik metotlar bir işlemin gerçekleşebilmesi için nesne oluşturulmasına gerek olmayan durumlarda kullanılır.

```
class Static
{
  public static int Topla(params int[] dizi)
  {
    int toplam = 0;
    for (int i = 0; i < dizi.Length; i++)
    { toplam += dizi[i]; }
    return toplam;
  }

  public static int Cikar(int s1, int s2)
  { return s1 - s2; }
}
```

kullanımı

```
Console.WriteLine(Static.Topla(1, 2, 3, 4, 5, 6, 7, 8, 9));
int fark = Static.Cikar(5, 4);
Console.WriteLine(fark);
```



- **Kalıtım (Inheritance)**
Kalıtım bir sınıfın bir üst sınıftan miras almasına denir. Adından da anlaşılacağı üzere üst sınıfın bütün özellikleri miras alan sınıfa aktarılır.

- **Sanal (Virtual) Metotlar**
Sanal metotlar ana sınıf içerisinde bildirilmiş ve miras alınan sınıfta tekrar bildirilebilen sınıflardır. Sanal metotlar kullanılarak nesne yönelimli programlamanın ilkesi olan çok biçimlilik (polymorphism) uygulanmış olur. Sanal metotları bildirmek için virtual anahtar sözcüğünü kullanırız, miras alan sınıfta ise override anahtar sözcüğünü kullanırız. (virtual – override)

```
class Canli
{
  public virtual void Beslenme()
  { }
}

class Insan : Canli
{
  public override void Beslenme()
  { base.Beslenme(); Console.WriteLine("Ezildi.") }
}
```

- **Sealed Sınıflar**
Eğer bir sınıfın türetilme yapılmasını istemiyorsak bu sınıfı sealed anahtar sözcüğü ile beraber tanımlarız.

```
sealed class Canli
{
  public void Beslenme()
  { }
}
```

- **Interface (Arayüz)**
Interface'ler kendisini uygulayan sınıfların kesin olarak içereceği metotları, özellikleri bildirirler. Interface'leri genellikle yazacağımız kodlara rehber olması amacıyla hazırlarız. Özellikle birden fazla programcı tarafından geliştirilen uygulamalarda ekibe büyük fayda sağlarlar.

```
interface IDatabaseOperations
{
  void add();
  void delete();
  void update();
  void getById();
}

class DatabaseOperations : IDatabaseOperations
{
  public void add()
  { }
  public void delete()
  { }
  public void getById()
  { }
  public void update()
  { }
}
```

- https://www.youtube.com/watch?v=YaVNGTJABkg
- https://www.youtube.com/watch?v=D1R5zQ4RDLw&t=3s

OOP (Nesne Yönelimli Programlama) Temel Prensipleri
1. Encapsulation (Kapsüllemek)
2. Inheritance (Kalıtım)
3. Polymorphism (Çok biçimlilik)

((Encapsulation(Kapsülleme))) : eğer sınıfımız içindeki bir alanın(field'ın) değişkenlerin herbirinin erişimini denetim altına almak istediğimizde yada kısıtlamak istediğimizde 'kapsülleme' işlemi yaparız.

((Constructor(Yapıcı Method))) : yazsakta yazmasakta program compile edilirken ilk code'lar içerisine eklenen özel bir method'dur. Bir nesnenin örneğini aldığımızda tetiklenir. Sınıfla adı birebir aynı bir geriye dönüş tipi bulunmamaktadır. İçerisine parametre alabilir veya almayabilir.(overloading)

((Inheritance(Kalıtım))) : bir sınıfın kalıtsal özelliklerinin, başka bir sınıfa akratılması(devredilmesi) anlamına geliyor.

Abstraction(soyutlama) :
((abstract class)) : Eğer bir class'ın türetilmesini(örneğini) oluşturmak istemiyorsak yani (new'lemek) istemediğimiz zaman kullanılır.
Kensisinden kalıtımı(inheritance) yapılabilen fakat örneği(Instance) alınamayan sınıflar tanımlamak için kullanılır.
Abstract bir sınıf içerisinde abstract metotlarda geliştirebiliriz.
Abstract olmayan bir sınıfta kesinlikle abstract metot yazılmaz.

((sealed class)) : Kalıtım yapılmasını istemediğimiz ama türetiletek kullanılmasını isteğimiz 'class'larımıza' sealed keyword'ü ile kullanırız. Kensisinden kalıtımı yapılmayan fakat örnek alınarak kullanılabilen sınıflar tanımlamak için kullanılır.

((polymorphism)) : Absctract olmayan bir metot olduğunda ve abstract bir sınıfın içinde değiliz ama polymorphism'i kullanmak istiyoruz. Bunun için : 'virtual' keyword'ünü kullanırız.
((virtual method)) : Abstract olmayan bir sınıfta polymorphism'e temel sağlamak amaçlı, virtual method oluşturulabilir.

C# Generics : Generic'ler tasarlandığımız interface, class, metod yada parametrelerin (argümanların) belirli bir tip için değil bir şablon yapısına uyan her tip için çalışmasını sağlayan bir yapıdır.
Generic sınıfların normal sınıflardan farkı, kullanılacak olan tiplerin sınıf tanımlama aşamasında belirtilmesidir