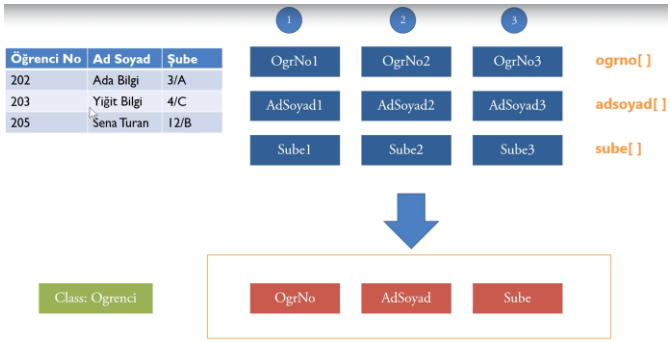
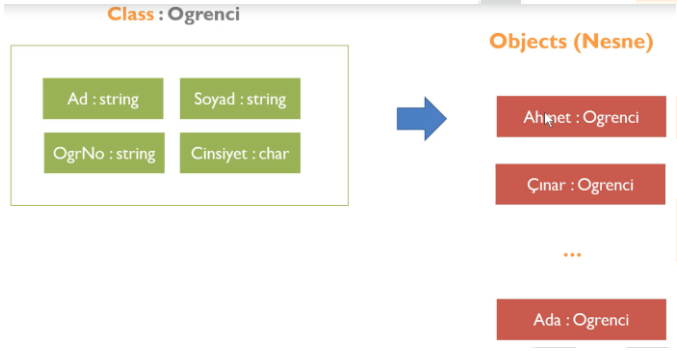
C# Nesne Yönelimli Programlama

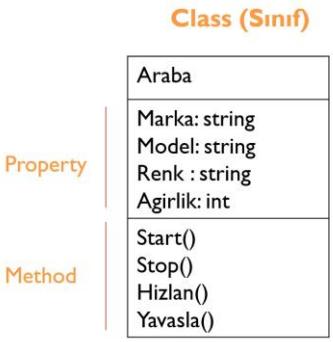
Class



- Biz normal olarak bir öğrenci listesi yaparsak ve isim, okul no ve sube bilgileri olunca farklı listelerde tutabilir. Lakin işler büyüyünce biraz karışıklık olacaktır.
- Aşağı kısımdaki gibi bir class oluşturarak daha kolay kontrol edilebilir bir yapı tasarlayabiliriz.



- Bir class yapısından Objects(nesne) oluşturulmaktadır.



- Class'da bir tek nesne tanımlamanın yanında bu nesnelere methodlar tanımlıyoruz ve bu metotlar sayesinde belli işlemleri bu metotlar içerisinde yapılabilirmektedir.

```
7   class Ogrenci
8   {
9       public int OgrNo { get; set; }
10      public string Ad { get; set; }
11      public string Sube { get; set; }
12  }
13
14  class Program
15  {
16      static void Main(string[] args)
17      {
18          Ogrenci ogr1 = new Ogrenci();
19          ogr1.OgrNo = 100;
20          ogr1.Ad = "Cinar";
21          ogr1.Sube ="10A";
22
23          Console.WriteLine($"no: {ogr1.OgrNo} ad: {ogr1.Ad} sube: {ogr1.Sube}");
24
25          //Tanimlama 2
26
27          Ogrenci ogr2 = new Ogrenci()
28          {
29              OgrNo = 200,
30              Ad="Ada",
31              Sube="10B"
32          };
33
34          Console.WriteLine($"no: {ogr2.OgrNo} ad: {ogr2.Ad} sube: {ogr2.Sube}");
35
36          Console.WriteLine("*************************");
37
38          Ogrenci[] ogrenciler = new Ogrenci[3];
39
40          ogrenciler[0] = ogr1;
41          ogrenciler[1] = ogr2;
42          for (int i = 0; i < ogrenciler.Length; i++)
43          {
44              Console.WriteLine($"no: {ogrenciler[i].OgrNo} ad: {ogrenciler[i].Ad} sube: {ogrenciler[i].Sube}");
45          }
46      }
47  }
```

- İlk olarak 7. Satırda class yazıp class ismini tanımlıyoruz.
- Daha sonra public yazıp property(değişken olarak da düşünülebilir) tanımlıyoruz. Süslü içerisindeki get ve set olmalı.
- 18. Satırda ogr1 isimli Objects(Nesne) tanımlıyoruz.
- 19-21. Satırlarda bu öğrencinin özelliklerini giriyoruz.
- 23. Satırda oluşturduğumuz ogr1 in adı vs yazdırıyoruz.
- 27. Satırda ogr1 kısmına göre daha kolay Objects (nesne) tanımlaması yapıyoruz.
- 38. Satırda Ogrenci dizisi oluşturdum ve 40-41. Satırlarda elemanlarını girdim.
- 42. Satırda for döngüsü ile elemanları yazdırdım.

Class Uygulama

```
7   class Product
8   {
9       public string Name { get; set; }
10      public double Price { get; set; }
11      public string Description { get; set; }
12  }
13
14  class Program
15  {
16      static void Main(string[] args)
17      {
18          // Product class -> name, price, description
19          // Sinirsiz sayida urun bilgisini alip bir dizi icinde saklayiniz.
20          // urun adetini kullanici belirtsin.
21          // Eklenen urunler listelensin.
22
23          Console.Write("adet: ");
24          int adet = int.Parse(Console.ReadLine());
25
26          Product[] products = new Product[adet];
27
28          int i=0;
29          Product prd;
30
31          do
32          {
33              prd = new Product();
34
35              Console.Write("urun adi: ");
36              prd.Name = Console.ReadLine();
37
38              Console.Write("urun fiyat: ");
39              prd.Price = double.Parse(Console.ReadLine());
40
41              Console.Write("aciklama: ");
42              prd.Description = Console.ReadLine();
43
44              products[i] = prd;
45              i++;
46
47          } while (adet>i);
```

```
51          foreach (var urun in products)
52          {
53              Console.WriteLine($"urun adi: {urun.Name} urun fiyat: {urun.Price} aciklama: {urun.Description}");
54          }
```

- 9. Satırda class tanımlaması yapıldı.
- Genel anlamda diğer kullanımları önceki notlarda anlattık.
- 51. Satırda foreach döngüsü örneği bulunmakta. Bu döngü ile products listesi içerisindeki elemanları tek tek urun içerisine atama işlemi yapar. Örnek ["bilal", "mehmet", "Ahmet"] var mesela ilk olarak bilal, sonra Mehmet sonra Ahmet değerini teker teker urun'e atar ve listenin bütün elemanlarını dolaşır.
- Böylelikle 53. Satırda product[i] yerine urun diyebilirdik. Çünkü product[i] elemanı ürüne eşit oldu.

Metotlar

```
3 references
class Person
{
    4 references
    public string Name { get; set; }
    4 references
    public int Year { get; set; }
    3 references
    public string Intro()
    {
        return $"isim: {this.Name} yaş: {this.CalculateAge()}";
    }
    1 reference
    public int CalculateAge()
    {
        return DateTime.Now.Year - this.Year;
    }
}
0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        // Class => object (nesne)
        // Ogrenci => ogr1,ogr2

        var p1 = new Person { Name = "Ada", Year = 2012 };
        var p2 = new Person { Name = "Yiğit", Year = 2010 };
        var p3 = new Person { Name = "Sena", Year = 1999 };

        Console.WriteLine(p1.Intro());
        Console.WriteLine(p2.Intro());
        Console.WriteLine(p3.Intro());
    }
}
```

- 9. Ve 13. Satırlarda metotlar tanımladık ve bu metotlar bir işlevi yerine getirmektedir.
- Intro metodu isim ve yaş hesaplamasını yapmaktadır, yaş hesaplamasını yaparken alttaki metodu çağırmaktadır. Matot çağrılırken metotismi() şeklinde çağrılır.
- Normalde nesne oluştururken p1, ogr1 gibi isim verip daha sonra ogr1.name çağırıyorduk. Class içerisinde daha nesne tanımlanmadığı ve oluşturulacak tüm nesneler için aynı özelliği tanımlaması için ogr veya p1 yerine this kullanılmaktadır.
- Return ile metoda bir değer dönderilmektedir. Örnek 13. Satırdaki metotda yaş hesaplaması yapıldı ve 20 çıktı. Bu metoda geri gönderilir ve 11. Satırın son kısmında oraya 20 girer.
- 25-27 Satırlarda nesne üretilmektedir.
- 29-31. Satırlarda bu öğrencilerin bilgileri yazılmaktadır. P1.Intro() dediğimiz zaman 9. Satırdaki metoda gider ve this yerlerine p1 koyup işlemi geri gönderir.
- Nesne oluştururken var da kullanılabilir, Person nesnesi oluşturacaktır zaten.

Metotlar Uygulama

```
class Araba
{
    public string Marka { get; set; }
    public string Model { get; set; }
    public string Renk { get; set; }
    public bool Otomatik { get; set; }

    public void Start()
    {
        Console.WriteLine($"{this.Marka} {this.Model} çalıştırıldı.");
    }

    public void Stop()
    {
        Console.WriteLine($"{this.Marka} {this.Model} stop edildi.");
    }

    public void Yavasla()
    {
        Console.WriteLine($"{this.Marka} {this.Model}a yavaşlıyor");
    }

    public void Hizlan()
    {
        Console.WriteLine($"{this.Marka} {this.Model} hızlanıyor");
    }
```

```
    public void Menu()
    {
        string komut = "";

        do
        {
            Console.WriteLine("1-Start 2-Hızlan 3-Yavaşla 4-Stop Çıkış: Q");
            Console.Write("Seçiminiz: ");
            komut = Console.ReadLine();

            switch (komut)
            {
                case "1":
                    this.Start();
                    break;
                case "2":
                    this.Hizlan();
                    break;
                case "3":
                    this.Yavasla();
                    break;
                case "4":
                    this.Stop();
                    break;
                default:
                    Console.WriteLine("uygulamadan çıkıldı.");
                    break;
            }

        } while (komut != "Q");
```

```
class Program
{
    static void Main(string[] args)
    {
        var opel = new Araba();
        opel.Marka = "Opel";
        opel.Model = "Astra";
        opel.Renk = "Beyaz";
        opel.Otomatik = true;

        // opel.Start();
        // opel.Hizlan();
        // opel.Yavasla();
        // opel.Stop();

        var mazda = new Araba();
        mazda.Marka="Mazda";
        mazda.Model="CX3";
        mazda.Renk="Kırmızı";
        mazda.Otomatik=true;

        opel.Menu();
        // mazda.Menu();
    }
}
```

- 8-11. Satırlarda propertyler tanımlandı. (Araba ortak özellikleri)
- 13-62 arasında metotlar tanımlandı, 33. Satırda menü hazırlandı.
- 71-90 arasında opel ve mazda nesneleri oluşturulup metotlar çağrıldı.

Metot Parametreleri

```
67      class Islem
68      {
69          // 1. Durum
70          public int Toplama(int x, int y=0, int z=0)
71          {
72              Console.WriteLine("x " + x);
73              Console.WriteLine("y " + y);
74              Console.WriteLine("z " + z);
75
76              return x + y + z;
77          }
78
79
80          // 2. Durum
81          public int Toplama(params int[] sayilar)
82          {
83              int toplam=0;
84              foreach (var sayi in sayilar)
85              {
86                  toplam += sayi;
87              }
88              return toplam;
89          }
90      }
```

- İşlem isminde bir class var ve 70. Satırdaki gibi bir metot var. Bu metoda () içerisinde parametre gönderebiliriz. Yani bu metotları sonradan dışarıdan tanımlayabiliriz.
- 1. Durumda 3 sayı toplanırken, 2. Durumda istenilen sayida sayi toplama işlemi yapılabilmektedir.
- 70. Satırda int y=0 olarak tanımlama durumu y ye varsayılan değeri 0 verildi. Eğer kullanıcı y'ye değer vermezse 0 kabul edilir, verirse girmiş olduğu değer kabul edilir.

```
96          var islem = new Islem();
97
98          // 1. Durum
99          Console.WriteLine(islem.Toplama(10, 20, 30));
100         Console.WriteLine(islem.Toplama(y: 20, z: 30, x: 10)); // Named
101         Console.WriteLine(islem.Toplama(10,20)); // default
102
103
104         // 2. Durum
105         Console.WriteLine(islem.Toplama(10));
106         Console.WriteLine(islem.Toplama(10,20));
107         Console.WriteLine(islem.Toplama(10,20,30));
108         Console.WriteLine(islem.Toplama(10,20,30,40));
```

- Şimdi 1. Durumda normal olarak metot çağrıldığında örnek Toplama(10,20,30) sırası ile yukarıda tanımladığımız sırayla x,y,z değerlerini alır (99.satır). Eğer bu sıralamayı kendi isteğimizle yapmak istersek 100. Satırdaki gibi tanımlama yapmamız gerekir.
- 105-108 arasında tanımlanan işlemler ise classda 2. Durum kısmını kapsamaktadır. Yani istenilen sayının toplamı yapılabilir.

Aşırı Yüklenmiş Metotlar

```
88      class Islem
89      {
90          public int Toplama(int a, int b)
91          {
92              return a + b;
93          }
94          public int Toplama(int a, int b, int c)
95          {
96              return a + b + c;
97          }
98
99          public int Toplama(int a, int b, int c, int d)
100         {
101             return a + b + c + d;
102         }
103     }
104
105     class Program
106     {
107         static void Main(string[] args)
108         {
109             var islem = new Islem();
110
111             Console.WriteLine(islem.Toplama(10,20));
112             Console.WriteLine(islem.Toplama(10,20,30));
113             Console.WriteLine(islem.Toplama(10,20,30,40));
```

- Aynı isimde birden fazla metot normalde olamaz. Eğer bu kullanmak istersek aşırı yüklenmiş metot taktiği uygulanması gerekiyor.
- Bu taktik de ise metot isimleri aynı olabilir lakin parametreler farklı olması gerekmektedir. Buna aşırı yüklenmiş metotlar denir.

Yapıcı Metotlar

```
6       class Araba
7       {
8           public Araba()
9           {
10              this.MaxHiz = 180;
11              Console.WriteLine("yapici metot Calistirildi.");
12          }
13          public Araba(int maxhiz)
14          {
15              this.MaxHiz = maxhiz;
16          }
17
18          public Araba(string marka,string model,string renk,bool otomatik,int maxhiz)
19          {
20              this.Marka = marka;
21              this.Model = model;
22              this.Renk = renk;
23              this.Otomatik = otomatik;
24              this.MaxHiz = maxhiz;
25          }
```

- Bu şekilde yapıcı metotlar oluşturulabilir. Yapıcı metotlar class ismi ile aynı isimde olur. Bu yapıcı metotlar nesne oluşturulurken direk çağrılır. Normal metotlar ise manuel çağrılması gerekir. Python daki __init__ yapısı olarak düşünülebilir.
- Tabi bu metotlar da aşırı yüklenmiş özelliği ile oluşturulabilir.

```
109         static void Main(string[] args)
110         {
111             var opel = new Araba(200);
112             Console.WriteLine(opel.MaxHiz);
113             opel.Marka = "Opel";
114             opel.Model = "Astra";
115             opel.Renk = "Beyaz";
116             opel.Otomatik = true;
117
118             var mazda = new Araba("Mazda","CX3","Kirmizi",true,220);
119
120             Console.WriteLine(mazda.Marka);
121             Console.WriteLine(mazda.Model);
122             Console.WriteLine(mazda.Renk);
123             Console.WriteLine(mazda.Otomatik);
124             Console.WriteLine(mazda.MaxHiz);
```

- 111. Satırdaki nesne tanımlaması 13. Satırdaki metot ile tanımlanmıştır. Max hız girilmiştir daha sonra diğer özellikler doldurulmuştur.
- 118. Satırdaki nesne ise 18. Satırdaki matot ile doldurulmuştur. Diğerine kıyasla bütün özellikler parametre olarak yazılmıştır. Lakin 20-24. Satırdaki gibi tanımlama yapılıyor olması gerekmektedir.

Yapıcı Metotlar Uygulama

```
class Comment
{
    public int CommentId { get; set; }
    public string Text { get; set; }
}

class Product
{
    public Product()
    {
        this.ProductId = (new Random()).Next(11111,99999);
        this.Comments = new Comment[3];
    }

    public Product(int productId):this()
    {
        this.ProductId = productId;
    }

    public Product(int productId,string name,double price,bool isApproved):this(productId)
    {
        this.Name = name;
        this.Price=price;
        this.IsApproved=isApproved;
    }


    public int ProductId { get; set; }
    public string Name { get; set; }
    public double Price { get; set; }
    public bool IsApproved { get; set; }
    public Comment[] Comments { get; set; }
}
```

```
class Program
{
    static void Main(string[] args)
    {
        var c1 = new Comment { CommentId=1,Text="Güzel telefon"};
        var p1 = new Product();

        p1.Comments[0] = c1;

        Console.WriteLine(p1.ProductId);
        Console.WriteLine(p1.Name);
        Console.WriteLine(p1.Price);
        Console.WriteLine(p1.IsApproved);
        Console.WriteLine(p1.Comments[0].Text);

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

        var p2 = new Product(1213);

        p2.Comments[0] = c1;

        Console.WriteLine(p2.ProductId);
        Console.WriteLine(p2.Name);
        Console.WriteLine(p2.Price);
        Console.WriteLine(p2.IsApproved);
        Console.WriteLine(p2.Comments[0].Text);

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

        var p3 = new Product(1231,"samsung s7",3000,true);

        Console.WriteLine(p3.ProductId);
        Console.WriteLine(p3.Name);
        Console.WriteLine(p3.Price);
        Console.WriteLine(p3.IsApproved);

        p3.Comments[0] = c1;
        Console.WriteLine(p3.Comments[0].Text);

    }
}
```

- Comment adındaki class yorumları, product adındaki class ürün bilgilerini tutmaktadır.
- Commnet de yorum id ve yorum yazısı bulunmaktadır.
- Product içerisinde 3 tane yapıcı metot yazdık, direk Product() şeklinde veya parametre göndererek nesne oluşturulabilir.
- 14. Satırdaki yapıcı metotda commnets listesi tanımlaması yapıldı yani yorumlar dizisi tanımlandı ve varsayılan 3 tane yorum olsun dedik. (Yorum sonra tanımlanacak.)
- 25 ve 20. Satırlardaki metot sonlarında .this yapısı bulunmaktadır. Bu metotlarda yorum için dizi tanımlaması yapılmadı (17. Satır). 20. Satırdaki .this diyerek 14. Metot çağrılmış oldu aslında. 25. Satırda ise direk 1.'yi çağıramadığımız için (productid random atıyor) 20. Satırdaki metot çağrılıp sonra 14. Satırdaki metot çağrılır.
- 33-37 arasında değişkenleri tanımlamasını yaptık.
- Program classı içerisinde main kodlarımız bulunmaktadır.
- 44. Satırda bir tane yorum tanımlandı. (max 3 tane tanımlayabilirim). Eğer kaç tane tanımlayacağınızı bilmediğiniz zaman ileride diziler yerine listeler kullanılacak.
- 47,59,76. Satırda yorum dizisine tanımladığımız yorumu atadık.

Properties

```
class Product
{
    private string _name;
    public string Name {
        get
        {
            return _name;
        }
        set
        {
            if(!string.IsNullOrEmpty(value))
            {
                _name = value;
            }
            else
            {
                throw new Exception("name alanı zorunlu.");
            }
        }
    }
    private double _price;
    public double Price
    {
        get
        {
            return _price;
        }
        set {
            if(value<0)
                _price=0;
            else
                _price=value;
        }
    }

    public bool IsApproved { get; }
}
```

```
class Program
{
    static void Main(string[] args)
    {
        var p = new Product();
        p.Name="Samsung S9";
        p.Price=2000;

        Console.WriteLine(p.Price);
        Console.WriteLine(p.Name);
    }
}
```

- 8. Satırda private tipinde bir değişken tanımlanmıştır. Bu değişkenler yanlızca ilgili class/metot içerisinde tanımlıdır. (_değişken) şeklinde tanımlanan değişkenler dışarıdan erişilmesi engellenmektedir. Örnek üniversite classında akademisyen maaşlarını dışarıdan erişilmesi bu şekilde engellenir.
- 9. Satır ile name değişkenine dışarıdan erişilmesi için kod yazılmıştır. Get ile bu değişkeni getirebilirken, set ile bu değişken güncellenmektedir.
- 16. Satırda IsNullOrEmpty ile boş değer olması sorgulanmaktadır.
- 22. Satırda hata fırlatılmaktadır. İlerleyen derslerde tekrar konusu işlenecek. Şimdilik Consola hata yazılmış gibi düşünülebilir.
- 26-27. Satırlarda name değişkenine yapılan aynı işlemler yapıldı.
- 41. Satırda eğer bir değişken okunsun ama değiştirilmesin istiyorsanız get bırakılıp set silinmektedir.

Uygulama 1 (Soru (Question) Sınıfı)

```
6   class Question
7   {
8       public Question(string text,string[] choices,string answer)
9       {
10          this.Text = text;
11          this.Choices = choices;
12          this.Answer = answer;
13      }
14      public string Text { get; set; }
15      public string[] Choices { get; set; }
16      public string Answer { get; set; }
17
18      public bool checkAnswer(string answer)
19      {
20          return this.Answer.ToLower() == answer.ToLower();
21      }
22  }
23
24  class Program
25  {
26      static void Main(string[] args)
27      {
28          // OOP: Quiz Uygulaması
29          var q1  = new Question("En iyi programlama dili hangisidir?",new string[]{"Python","C#","Java","C++"},"C#");
30          var q2  = new Question("En populer programlama dili hangisidir?",new string[]{"C#","Python","Java","C++"},"C#");
31          var q3  = new Question("En cok kazandiran programlama dili hangisidir?",new string[]{"C#","Java","Python","C++"},"C#");
32
33          var questions = new Question[]{q1,q1,q3};
34
35          int index=1;
36          foreach (var q in questions)
37          {
38              Console.WriteLine($"soru {index}: {q.Text}");
39              index++;
40
41              foreach (var c in q.Choices)
42              {
43                  Console.WriteLine($"-{c}");
44              }
45
46              Console.Write("cevap: ");
47              var cevap = Console.ReadLine();
48              Console.WriteLine(q.checkAnswer(cevap));
49          }
50      }
51  }
```

- Yukarıda bir quiz uygulaması yapılmıştır. Text: soru, Choices şıklar, Answer verilen cevaptır.
- 18. Satırda bir soruya verilen cevap kontrolü yapılmaktadır. Metoda gelen answer'in yazısını küçük harfe çevirip, cevabıda küçük harfe çevirip kontrol yapar.
- 29-31 sorular,şıklar,cevap tanımlanmıştır.
- 33. Satırda quiz dizisi içerisine sorular tanımlamıştır.
- 36. Satır ile consol uygulaması yapıldı. Soru geliyor, şıklar yazıyor daha sonra cevap veriliyor, cevaba göre True False dönmektedir.
- 33. Satırda q1,q2,q3 olmalıdır.

```
24      class Quiz
25      {
26          public Quiz(Question[] questions)
27          {
28              this.Questions = questions;
29              this.QuestionIndex = 0;
30          }
31          private Question[] Questions { get; set; }
32          public int QuestionIndex { get; set; }
33
34          public Question GetQuestion()
35          {
36              return this.Questions[this.QuestionIndex];
37          }
38
39          public void DisplayQuestion()
40          {
41              var question = this.GetQuestion();
42              Console.WriteLine($"soru {this.QuestionIndex + 1}: {question.Text}");
43
44              foreach (var c in question.Choices)
45              {
46                  Console.WriteLine($"-{c}");
47              }
48
49              Console.Write("cevap: ");
50              var cevap = Console.ReadLine();
51              this.Guess(cevap);
52          }
53
54          public void Guess(string answer)
55          {
56              var question = this.GetQuestion();
57              Console.WriteLine(question.checkAnswer(answer)); // skor
58              this.QuestionIndex++;
59
60              if (this.Questions.Length == this.QuestionIndex)
61              {
62                  // skor
63                  return;
64              }
65              else
66              {
67                  this.DisplayQuestion();
68              }
69          }
```

- Questions dizisi ile soruları, QuestionIndex ile soruların index numarası (1.2.3. soru gibi) tanımlandı. İndex numarasına göre soruların gösterim işlemini yapacağız. Rastgele soruların gösterilmesi istenirse .Random kullanabilir.
- 31. Satırda sorular private şeklinde tanımlandı.
- 34. Satırdaki metot ile bu soruları getirme işlemi yapılmaktadır. [] içerisinde index'e göre geldiğine dikkat edelim. Örnek [1] içerisinden gelecek soru aslında 2. Soru. [0] 1. Soruya takabül eder.
- 39. Satırdaki metot, sorular gösterme işlevi yapmaktadır. 41. Satır ile soru getirilmektedir. 42. Satırda soru yazdırılıp, 44. Satırda şıklar gelmektedir. 49-50. Satırda kullanıcıdan cevap alır ve Guess isminde metot'a(53.satır) cevabı göndermektedir.
- 53. Satırdaki Guess metodu, gelen cevabı kontrol edip ekrana yazdırır, sonraki soruya geçer. 55. Satır ile quiz'i getirdik, böylelikle 56. Satırda sorunun kontrolü yapılıp ekrana yazdırıldı (bir sonraki dersde skor hesaplanacak buraya değişken vercez.) 57. Satır ile sorunun index'ini arttırdık, 59. Satırdaki if ile eğerki quiz içerisinde soru yoksa skor yazdıracağız, eğer soru bitmemişse 66. Satır ile soru getirmeye devam eder.
- 66. Satırda ekranda gösterim fonksiyonu çağrılmaktadır. Aslında burada **Recursion (Özyineleme)** li fonksiyon kullanılmıştır. Anlaşılmadıysa internette araştırabilirsiniz.

Uygulama 2 (Quiz Sınıfı)

```
6       class Question
7       {
8           public Question(string text, string[] choices, string answer)
9           {
10              this.Text = text;
11              this.Choices = choices;
12              this.Answer = answer;
13          }
14          public string Text { get; set; }
15          public string[] Choices { get; set; }
16          public string Answer { get; set; }
17
18          public bool checkAnswer(string answer)
19          {
20              return this.Answer.ToLower() == answer.ToLower();
21          }
22      }
```

- Soru classında bir değişiklik yapılmadı.
- Şimdi bir quiz sınıfı tanımlayalım ve soru değiştirme, cevap gibi işlevleri o class içerisinden tanımlayıp sadece class çağırma işlemi yapalim.

```
70
71      0 references
        class Program
72      {
            0 references
73          static void Main(string[] args)
74          {
75              // OOP: Quiz Uygulaması
76
77              var q1 = new Question("En iyi programlama dili hangisidir?", new string[] { "Python", "C#", "Java", "C++" }, "C#");
78              var q2 = new Question("En populer programlama dili hangisidir?", new string[] { "C#", "Python", "Java", "C++" }, "C#");
79              var q3 = new Question("En cok kazandiran programlama dili hangisidir?", new string[] { "C#", "Java", "Python", "C++" }, "C#");
80
81              var questions = new Question[] { q1, q2, q3 };
82              var quiz = new Quiz(questions);
83
84              quiz.DisplayQuestion();
85
86          }
87      }
88  }
89
```

- 77-79. Satırlar sorular tanımlandı.
- 81. Satırda sorular dizisi oluşturuldu.
- 82. Satırda sorular dizisi Quiz Classına verildi.
- 84. Satırda soruları gösterecek metot çağrıldı. İşlem bu kadardı.

Uygulama 3 (Skor Bilgilerinin Hazırlanması)

```
class Quiz
{
    1 reference
    public Quiz(Question[] questions)
    {
        this.Questions = questions;
        this.QuestionIndex = 0;
        this.Score = 0;
    }
    4 references
    private Question[] Questions { get; set; }
    6 references
    private int QuestionIndex { get; set; }
    3 references
    private int Score { get; set; }

    2 references
    private Question GetQuestion()
    {
        return this.Questions[this.QuestionIndex];
    }

    2 references
    public void DisplayQuestion()
    {
        var question = this.GetQuestion();
        this.DisplayProgress();
        Console.WriteLine($"soru {this.QuestionIndex + 1}: {question.Text}");

        foreach (var c in question.Choices)
        {
            Console.WriteLine($"-{c}");
        }

        Console.Write("cevap: ");
        var cevap = Console.ReadLine();
        this.Guess(cevap);
    }
```

```
    1 reference
    private void Guess(string answer)
    {
        var question = this.GetQuestion();
        if (question.checkAnswer(answer))
            this.Score++;
        this.QuestionIndex++;

        if (this.Questions.Length == this.QuestionIndex)
        {
            this.DisplayScore();
        }
        else
        {
            this.DisplayQuestion();
        }

    }
    1 reference
    private void DisplayScore()
    {
        Console.WriteLine($"Score: {this.Score}");
    }

    1 reference
    private void DisplayProgress()
    {
        int totalQuestion = this.Questions.Length;
        int questionNumber = this.QuestionIndex+1;

        if (totalQuestion>=questionNumber)
            Console.WriteLine($"Question {questionNumber} of {totalQuestion}");
    }

}
```

- 44. Satır eklendi, 79. Satırdaki metodu çağırmaktadır. Bu metot ile kullanıcı kaçıncı soruda olduğunu görebilmektedir.
- 30. Satırda ve 34. Satırda score değişkeni tanımlandı.
- 60. Satırda eğer cevap doğru verildiyse 61. Satırda skor 1 artmaktadır. (Değişmeden önce console.write ile doğru veya yanlış yazmaktaydı.)
- 64. Satır ile eğer quiz içerisindeki sorular bitmişse skoru yazdıracak metot çağrılmaktadır (66.satır).
- 74. Satırdaki metot ile scor bilgileri yazılmaktadır.
- 79. Satırdaki metot ile kullanıcı kaçıncı soruda olduğunu görebilmektedir.

```
$ dotnet run
Question 1 of 3
soru 1: En iyi programlama dili hangisidir?
-Python
-C#
-Java
-C++
cevap: C#
Question 2 of 3
soru 2: En popüler programlama dili hangisidir?
-C#
-Python
-Java
-C++
cevap: Python
Question 3 of 3
soru 3: En çok kazandıran programlama dili hangisidir?
-C#
-Java
-Python
-C++
cevap: Java
Score: 1
```

Statik Members

```
class Student
{
    public string Name { get; set; }
    public int StudentNumber { get; set; }

    public static string School = "my school";
    public static string Address = "my school address";

    public Student(string name, int studentnumber)
    {
        this.Name = name;
        this.StudentNumber=studentnumber;
    }

    public void DisplayStudentDetails()
    {
        Console.WriteLine($"name: {this.Name} number: {this.StudentNumber}");
    }

    public static void DisplaySchoolDetails()
    {
        Console.WriteLine($"school name: {School} address: {Address}");
    }
}
```

- 11. Ve 12. Satırda statik değişken tanımlandı. Bu değişkenler genel olarak nesne üretme noktasında kullanılmamaktadır. Örnek öğrencilerin isimleri, numaraları değişebilir lakin okul ismi adresi aynı kalmaktadır. İşte bu noktada kullanılmaktadır.
- 25. Satırdaki metotda statik bir metot tanımlanmıştır. 27. Satırda statik ile tanımladığımız değişkenler ile nesne tanımlanmadığı için this kullanılmamaktadır.
- Şimdi statik class oluşturalım.

```
static class HelperMethods
{
    public static string KarakterDuzelt(string str)
    {
        return str
                .Replace("ö","o")
                .Replace("ü","u")
                .Replace("ç","c")
                .Replace(" ","-")
                .Replace("ğ","g");
    }
}
```

- Yukarıda statik class tanımlanmıştır. Statik classlar içerisinde bir işlem yapılması için kullanılır, nesne üretilemez.

```
class Program
{
    static void Main(string[] args)
    {
        var s1 = new Student("Çınar",100);
        var s2 = new Student("Sena",101);
        var s3 = new Student("Yiğit",102);

        Student.DisplaySchoolDetails();

        s1.DisplayStudentDetails();
        s2.DisplayStudentDetails();
        s3.DisplayStudentDetails();


        string str = "ölçme ve değerlendirme";
        var result = HelperMethods.KarakterDuzelt(str);
        Console.WriteLine(result);

    }
}
```

- 52. Satır Statik metot çağrılırken nesne üzerinden çağrılmaz. Örnek s1.DisplayStudentDetails demek yerine Student.DisplaySchoolDetails diyerek class ismi ile çağırdık.
- 59-61. Satırlardaki örnek statik class kullanımına örnektir.

Statik Members Uygulama

```
class Product
{
    public int ProductId { get; set; }
    public string ProductName { get; set; }
    public double Price { get; set; }
    public bool IsApproved { get; set; }
}

static class ProductManager
{
    static Product[] Products;

    static ProductManager()
    {
        Products = new Product[5];

        Products[0] = new Product { ProductId = 1, ProductName = "Iphone 5", Price = 2000, IsApproved = false };
        Products[1] = new Product { ProductId = 2, ProductName = "Iphone 6", Price = 3000, IsApproved = false };
        Products[2] = new Product { ProductId = 3, ProductName = "Iphone 7", Price = 4000, IsApproved = true };
        Products[3] = new Product { ProductId = 4, ProductName = "Iphone 8", Price = 5000, IsApproved = true };
        Products[4] = new Product { ProductId = 5, ProductName = "Iphone X", Price = 6000, IsApproved = true };
    }

    public static Product[] GetProducts()
    {
        return Products;
    }

    public static Product GetProductById(int id)
    {
        Product product = null;

        foreach (var p in Products)
        {
            if (p.ProductId == id)
            {
                product = p;
                break;
            }
        }

        return product;
    }

    public static Product GetProductByName(string name)
    {
        Product product = null;

        foreach (var p in Products)
        {
            if (p.ProductName.Contains(name.ToLower()))
            {
                product = p;
                break;
            }
        }
        return product;
    }

}

class Program
{
    static void Main(string[] args)
    {

        var product = ProductManager.GetProductById(2);
        Console.WriteLine($"name: {product.ProductName} price: {product.Price}");

        var products = ProductManager.GetProducts();
        foreach (var p in products)
        {
            Console.WriteLine($"name: {p.ProductName} price: {p.Price}");
        }

        var product = ProductManager.GetProductByName("Phone");
        Console.WriteLine($"name: {product.ProductName} price: {product.Price}");

    }
}
```

- Ürün ile uygulama yapılacaktır. ProductManager bir veri tabanı olarak düşünülebilir.
- 15. Satırda statik bir class tanımladık.
- 30. Satırdaki metot products listesini getirmektedir.
- 35. Satırdaki metot İd'ye göre ürün getirmektedir.
- 51. Satırdaki metot Name göre ürün getirmektedir. Lakin aynı isimde iki ürün varsa ilkini getirir.
- Statik class olduğu için veriler program içerisinde değil, class içerisinde tanımlandı. (Veri tabanı gibi)
- 73-74. Satırlarda id'ye göre arama yapıp yazdırdık.
- 76-80. Satırlarda bütün ürünleri ekranda yazdırdık.
- 82-83. Satırla isme göre arama yapıldı.

Kalıtım

```
class Person
{
    public string Name { get; set; }
    public string SurName { get; set; }

    public Person(string name, string surname)
    {
        this.Name = name;
        this.SurName = surname;
        Console.WriteLine("Person nesnesi oluşturuldu.");
    }

    public virtual void Intro()
    {
        Console.WriteLine($"name: {this.Name} Surname: {this.SurName}");
    }
}

class Student: Person
{
    public string StudentNumber { get; set; }
    public Student(string name, string surname,string studentnumber): base(name,surname)
    {
        this.StudentNumber = studentnumber;
        Console.WriteLine("Student nesnesi oluşturuldu.");
    }

    public override void Intro()
    {
        Console.WriteLine($"name: {this.Name} Surname: {this.SurName} Number: {this.StudentNumber}");
    }
}

class Teacher: Person
{
    public string Branch { get; set; }
    public Teacher(string name, string surname,string branch):base(name,surname)
    {
        this.Branch = branch;
    }

    public void Teach()
    {
        Console.WriteLine("I am teaching...");
    }

    public override void Intro()
    {
        Console.WriteLine($"name: {this.Name} Surname: {this.SurName} Branch: {this.Branch}");
    }
}

class Program
{
    static void Main(string[] args)
    {
        // Inheritance (Kalıtım): Miras Alma
        // Person => name, surname, age, eat(), drink(), run()
        // Student(Person) => studentnumber, study()
        // Teacher(Person) => branch, teach()

        var p = new Person("Ali","Yılmaz");
        var s = new Student("Çınar","Turan","120");
        var t = new Teacher("Sadık","Turan","Bilişim");

        t.Intro();
        t.Teach();
        p.Intro();
        s.Intro();
    }
}
```

- Kalıtım özelliği ile örnek bir üniversitede kişi diye bir class tanımlanır ve her akademisyen,öğrenci,personelin isim soyismi vardır. Ortak özellikleri daha sonra oluşturulan öğrenci veya akademisyen sınıfında tekrar tanımlamaya gerek yoktur. Kalıtım yolu ile aktarım mümkündür.
- 4. Satırdaki Person classı bizim anasınıfımızdır, 16. Satırda Intro isminde bir metot bulunmaktadır. Bu metot alt classlarda tanımlandığında alt sınıftaki Intro ezmesi için virtual yazılır.
- 22. Satırda Student:Person ile kalıtım gerçekleşir, 25.. satırda base içerisine kalıtım yolu ile alınacak isim ve soyismi aldık, öğrencilerin extra öğrenci numarası olduğundan 24. Satırda öğrenci numarası tanımlandı.
- 31. Satırda override ile Intro metodu yukarıdaki virtual Intro metodunu ezer ve öğrenci isim soyismin yanında okul numarasını da yazar.
- Kalıtımın çalışma prensibi örneğimizdeki Person classı (Ana Class) ve Student (Child Class) bulunmaktadır. Student sınıfı person sınıfından kalıtım ile aldığı içi bir student nesnesi oluşturulduğunda ilk olarak person sınıfı çalıştırılır sonra student sınıfı çalıştırılır.

Abstract Sınıflar

```
58      abstract class Shape
59      {
60          public int Width { get; set; }
61          public int Height { get; set; }
62
63          public Shape(int w, int h)
64          {
65              this.Width = w;
66              this.Height= h;
67          }
68
69          public int CalculateArea()
70          {
71              return this.Width * this.Height;
72          }
73          public abstract void Draw();
74      }
75
76      class Square: Shape
77      {
78
79          public Square(int w, int h):base(w,h)
80          {
81          }
82          public override void Draw()
83          {
84              Console.WriteLine("Draw a square");
85          }
86      }
```

```
87
88      class Rectangle: Shape
89      {
90          public Rectangle(int w, int h):base(w,h)
91          {
92          }
93          public override void Draw()
94          {
95              Console.WriteLine("Draw a rectangle");
96          }
97      }
98
99      class Program
100     {
101         static void Main(string[] args)
102         {
103             // Abstract Class: Soyut Sınıf
104
105             var shapes = new Shape[3];
106
107             shapes[0] = new Rectangle(10,15);
108             shapes[1] = new Square(15,15);
109             shapes[2] = new Rectangle(15,20);
110
111             foreach (var shape in shapes)
112             {
113                 shape.Draw();
114                 Console.WriteLine($"alan: {shape.CalculateArea()}");
115             }
116
117         }
118     }
```

- Abstract sınıflar ve metotlar ile kalıtım yolu ile alan alt sınıflar için belirli özellikleri doldurmasını zorunlu hale getirebiliriz.
- 58. Satırda abstract class tanımlandı ve 73. Satırdaki Draw metodu abstract olarak tanımlandı. Bu yüzden 76. Ve 88. Satırdaki sınıflar Draw metodunu kullanmak zorundalar.
- Abstract sınıflar içerisinde normal metotlar oluşturulabilir lakin abstract sınıflar nesne oluşturamaz. Alt sınıflar kullanabilirler. Örnek 114. Satırda CalculateArea metodu kullanıldı.

```
        1 reference
69      class Square: Shape
70      {
            2 references
71          public override void Draw()
72          {
73              base.Draw();
74              Console.WriteLine("Draw a square");
75          }
76      }
```

- Eğer üst sınıftaki bir metot çağrılmak istenirse, base ile çağırabilirsiniz. (73.satır).
- Ekstra bir bilgidir. Kodun orijinali yukarıdaki gibidir.

Interface

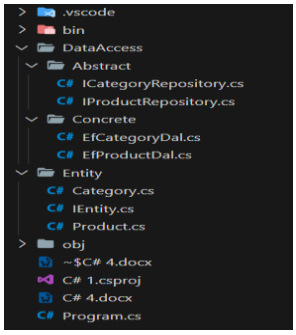
```
5       interface IKisi
6       {
7           string adSoyad {get;set;}
8           string adres {get;set;}
9           string departman { get; set; }
10          double maas { get; set; }
11
12      }
13      interface IPersonel
14      {
15          string departman { get; set; }
16          void bilgi();
17
18      }
19
20      class Yonetici : IPersonel,IKisi
21      {
22          public Yonetici(string _adsoyad,string _adres,string _departman)
23          {
24              this.adSoyad = _adsoyad;
25              this.adres = _adres;
26              this.departman =_departman;
27          }
28          public string adSoyad { get; set; }
29          public string adres { get; set; }
30          public string departman { get; set; }
31          public double maas{ get; set; }
32
33          public void bilgi()
34          {
35              Console.WriteLine($"{this.adSoyad} isimli personel {this.departman} bölümünde yöneticidir.");
36          }
37      }
```

```
39      class Isci: IPersonel,IKisi
40      {
41          public Isci(string _adsoyad,string _adres,string _departman)
42          {
43              this.adSoyad = _adsoyad;
44              this.adres = _adres;
45              this.departman =_departman;
46          }
47          public string adSoyad { get; set; }
48          public string adres { get; set; }
49          public string departman { get; set; }
50          public double maas { get; set; }
51
52          public void bilgi()
53          {
54              Console.WriteLine($"{this.adSoyad} isimli personel {this.departman} bölümünde işçidir.");
55          }
56      }
57
58      class Robot : IPersonel
59      {
60          public Robot(string _departman)
61          {
62              this.departman = _departman;
63          }
64          public string departman { get; set; }
65
66          public void bilgi()
67          {
68              Console.WriteLine($"{this.departman} bölümünde bir robot.");
69          }
70      }
```

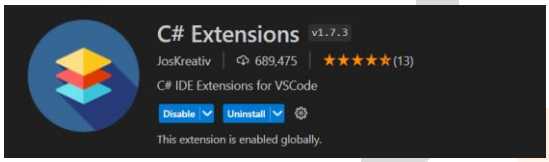
```
72      class Program
73      {
74          static void Main(string[] args)
75          {
76              // Interface
77              var personeller = new IPersonel[3];
78
79              personeller[0] = new Yonetici("ali yılmaz","istanbul","finans");
80              personeller[1] = new Isci("ahmet cengiz","kocaeli","üretim");
81              personeller[2] = new Robot("üretim");
82
83              foreach (var personel in personeller)
84              {
85                  personel.bilgi();
86              }
87          }
88      }
```

- İnterface bir yapı olarak düşünülebilir. Bir string yapısı gibi. Kullanıcılar bir nesne veya değişken tanımladıklarında, tip olarak seçili interface yapısını gösterirse interface içerisindeki yapıları zorunlu kılar.
- 5-13. Satırlarda iki tane interface tanımlanmıştır. “I” harfi ile başlaması zorunlu değil lakin yazılımcılar bu şekilde tanımların interface olduğunu anlar.
- Interface tanımlamalarında değişken tanımlarken (7-10. Satırlar) başında public vs yazılmaz.
- Kişi interface şirketteki insan personelleri temsil eder, Personel interface ise insan + robotları temsil etmektedir.
- 20. Satırda 2 tane interface kullanılmıştır. İnterface tanımlı değişkenlerde 28-31. Satırlardaki aynı değişkenler tanımlanmak zorundadır.

Interface Uygulama 1 (Repository Pattern)



- Dosyalarımız şuanda bu şekilde oluştururken klasör isimlerini kendiniz oluşturun. Daha sonra verdiğim eklenti ile sağ tık yaptığınıca C# class veya ınterface dosyası oluşturun.
- Abstract içerisindeki dosyalar ve Entity içerisindeki IEntity.cs dosyası interface dosyasıdır.



- Kurulacak eklenti yukarıdaki eklentidir.

Entity/IEntitiy.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace C__1.Entity
{
    1 reference
    public interface IEntity
    {
        1 reference
        int Id { get; set; }

    }
}
```

- Bu dosya ile category ve product dosyasının temel yapısı id tanımlandı. (Interface)
- Entity klasörü içerisindeki dosyalar ile veri taşıncak. Örnek product içerisinde sınıf tanımlanıp ilerleyen zamanda nesne oluşturup paketleyip göndercez. Burada nesnelerin ana tipleri.

Entity/Category.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace C__1.Entity
{
    6 references
    public class Category:IEntity
    {
        1 reference
        public int Id { get; set; }
        0 references
        public string Name { get; set; }
    }
}
```

- Bu dosya ile Category classının ayarlamasını yaptık.

Entity/Product.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace C__1.Entity
{
    8 references
    public class Product:IEntity
    {
        1 reference
        public int Id { get; set; }
        0 references
        public string Name { get; set; }
        0 references
        public double Price { get; set; }
    }
}
```

- Bu dosya ile product classının ayarlamasını yaptık.

DataAccess/Abstract/ICategoryRepository.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using C__1.Entity;

namespace C__1.DataAccess.Abstract
{
    1 reference
    public interface ICategoryRepository
    {
        1 reference
        Category GetById(int id);
        1 reference
        void Update(Category entity);
        1 reference
        void Create(Category entity);
        1 reference
        void Delete(int id);
    }
}
```

- DataAccess içerisinde veri tabanından veri çekme, gönderme gibi pek çok işlem bu klasör içerisinde yer alacaktır.
- Bu interface, **Category metotlarının yapıları** tanımlanmıştır.

DataAccess/Abstract/IProductRepository.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using C__1.Entity;

namespace C__1.DataAccess.Abstract
{
    1 reference
    public interface IProductRepository
    {
        1 reference
        Product GetById(int id);

        1 reference
        void Update(Product entity);

        1 reference
        void Create(Product entity);
        1 reference
        void Delete(int id);
        1 reference
        Product[] GetProductsByCategory(int id);
    }
}
```

- Bu interface, **Product metotlarının yapıları** tanımlanmıştır.

DataAccess/Concrete/EfCategoryDal.cs

```
using System;
using C__1.DataAccess.Abstract;
using C__1.Entity;

namespace C__1.DataAccess.Concrete
{
    0 references
    public class EfCategoryDal : ICategoryRepository
    {
        1 reference
        public void Create(Category entity)
        {
            throw new NotImplementedException();
        }

        1 reference
        public void Delete(int id)
        {
            throw new NotImplementedException();
        }

        1 reference
        public Category GetById(int id)
        {
            throw new NotImplementedException();
        }

        1 reference
        public void Update(Category entity)
        {
            throw new NotImplementedException();
        }
    }
}
```

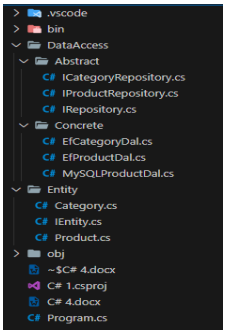
- Buraya interface de tanımlanan matotların içerisi doldurulmaktadır. (Category)

DataAccess/Concrete/EfProductDal.cs

```
using System;
using C__1.DataAccess.Abstract;
using C__1.Entity;
namespace C__1.DataAccess.Concrete
{
    0 references
    public class EfProductDal : IProductRepository
    {
        1 reference
        public void Create(Product entity)
        {
            throw new NotImplementedException();
        }

        1 reference
        public void Delete(int id)
        {
            throw new NotImplementedException();
        }

        1 reference
        public Product GetById(int id)
        {
            throw new NotImplementedException();
        }

        1 reference
        public Product[] GetProductsByCategory(int id)
        {
            throw new NotImplementedException();
        }

        1 reference
        public void Update(Product entity)
        {
            throw new NotImplementedException();
        }
    }
}
```

- Product metotları bu bölüm içerisinde doldurulmaktadır.
- Uygulama olduğu için genel yapıyı anlamaya çalışalım. Şuan veri tabanı ile bir çalışma işlemi söz konusu değildir. Sadece genel şema hazırlanmaktadır.

Generic Interface Uygulama 2 (Repository Pattern)



- Abstract / IRepository.cs ve Entity/MySQLProductdal.cs eklendi.

DataAccess/Abstract/IRepository.cs

```
namespace C__1.DataAccess.Abstract
{
    2 references
    public interface IRepository<TEntity>
    {
        4 references
        TEntity GetById(int id);
        4 references
        void Update(TEntity entity);
        6 references
        void Create(TEntity entity);
        4 references
        void Delete(int id);
    }
}
```

- IProductRepository ve ICategoryRepository içerisindeki ortak metotları buraya aldık. 3. Satırdaki TEntity yerine yazmamız yeterli. 5,6,7. Satırda TEntity yerine normalde Category veya Product yazmaktaydı.

DataAccess/Abstract/IProductRepository.cs

```
using C__1.Entity;

namespace C__1.DataAccess.Abstract
{
    5 references
    public interface IProductRepository: IRepository<Product>
    {
        3 references
        Product[] GetProductsByCategory(int id);
        3 references
        Product[] GetPopularProduct();
    }
}
```

DataAccess/Abstract/ICategoryRepository.cs

```
using C__1.Entity;

namespace C__1.DataAccess.Abstract
{
    1 reference
    public interface ICategoryRepository : IRepository<Category>
    {
        1 reference
        Category[] GetCategories();
    }
}
```

- IReposity dosyası içerisinde tanımladığımız metotları tekrar yazmak zorunda kalmadık. 5. Satırdaki gibi <> içerisine Category veya Product yerleştirilmesi yeterli. Yanlızca ekstra metot yazılır.

DataAccess/Concrete/MySQLProductDal.cs

```
namespace C__1.DataAccess.Concrete
{
    1 reference
    public class MySQLProductDal : IProductRepository
    {
    }
}
```

Quick Fix...
Implement interface
Implement (Enter to apply, Ctrl+Enter to preview)
Implement remaining members explicitly
Surround With...
Surround With: for loop
Surround With: While loop
Surround With: #if
Surround With: #region
Surround With Snippet...

```
using System;
using C__1.DataAccess.Abstract;
using C__1.Entity;

namespace C__1.DataAccess.Concrete
{
    1 reference
    public class MySQLProductDal : IProductRepository
    {
        2 references
        public void Create(Product entity)
        {
            Console.WriteLine("MySQLProduct - Create");

        }

        1 reference
        public void Delete(int id)
        {
            throw new NotImplementedException();
        }
    }
}
```

- 7. Satırdaki :IProductRepository üzerine çift tıklayıp, ampule tıkladığınızda ImpLement interface diyinde otomatik metotlar yerleşecektir.
- 9. Satırdaki metot üzerinde değişiklik yapıldı. (Create).

DataAccess/Concrete/EfProductDal.cs

```
using System;
using C__1.DataAccess.Abstract;
using C__1.Entity;
namespace C__1.DataAccess.Concrete
{
    0 references
    public class EfProductDal : IProductRepository
    {
        2 references
        public void Create(Product entity)
        {
            Console.WriteLine("EFProduct - Create");
        }

        1 reference
        public void Delete(int id)
        {
            throw new NotImplementedException();
        }
    }
}
```

- 8. Satırdaki metot değiştirilecek. (Create).

Program.cs

```
using System;
using C__1.DataAccess.Abstract;
using C__1.DataAccess.Concrete;
using C__1.Entity;

namespace C__1
{
    2 references
    class ProductManager : IProductRepository
    {
        2 references
        IProductRepository _repository;
        1 reference
        public ProductManager(IProductRepository repository)
        {
            _repository = repository;
        }

        3 references
        public void Create(Product entity)
        {
            _repository.Create(entity);
        }
    }
```

```
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            // var productDal = new EfProductDal();
            // var productDal = new MySQLProductDal();
            // productDal.Create(new Product());

            // var productDal = new ProductManager(new EfProductDal());
            var productDal = new ProductManager(new MySQLProductDal());
            productDal.Create(new Product());
        }
    }
}
```

- 8. Satırdaki class'ı tanımlamadan 50-53. Satırlardaki gibi kullanılabilir. MySQL teknolojisi veya Ef teknolojisi kullanılabilir. Ki ileriki derslerde daha anlaşılır olacaktır.
- 8.satırda Class ismini yazdıktan sonra IProductReposity ekleyip implement edin. Daha sonra 10-19. Satırları düzenleyin, 55-57. Satırlardaki gibi kullanabilirsiniz