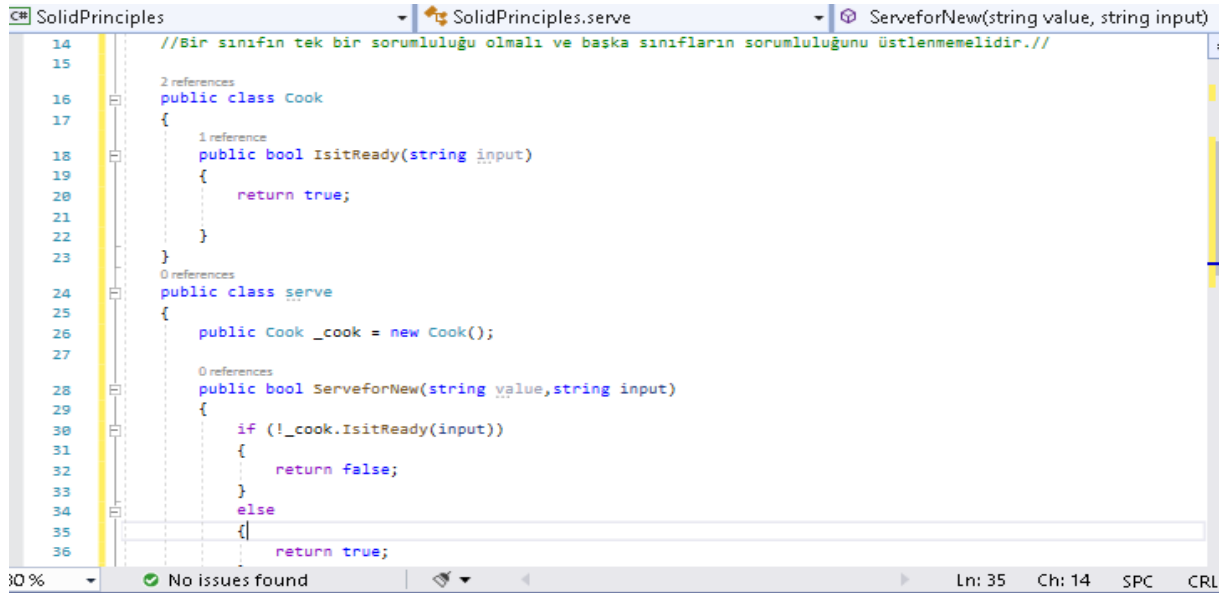


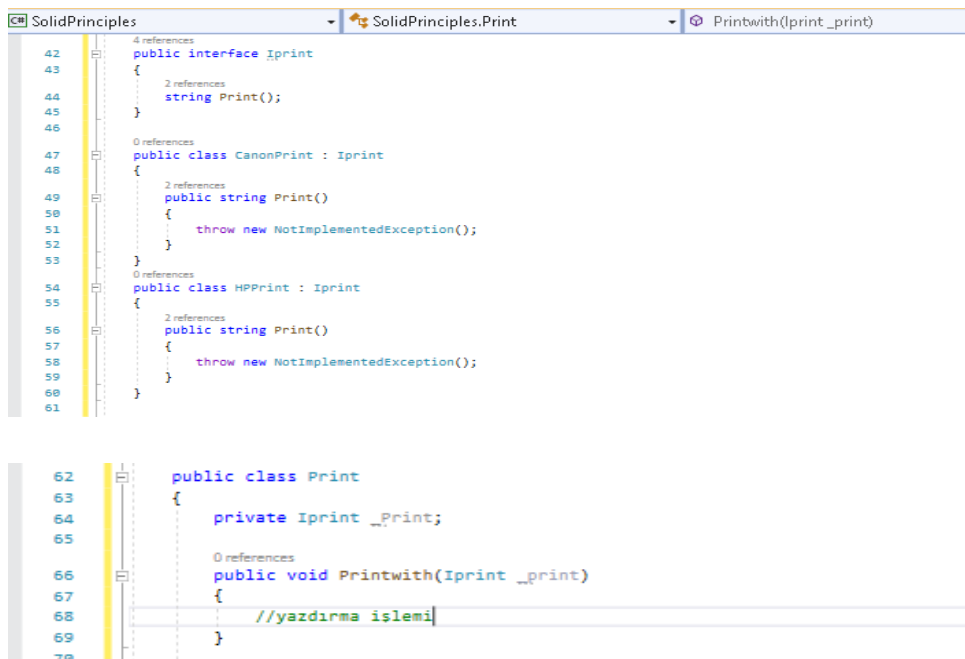
Solid Prensipleri

1. Bir sınıfın sorumluluğu sadece kendi görevi olmalıdır. Başka sınıfların sorumluluğunu üstlenmemelidir. Örneğin bir lokantada servisi yapan ve yemeği pişiren sınıflar ayrı ayrıdır. Aşağıda örnekle ifade edilmiştir.



```
//Bir sınıfın tek bir sorumluluğu olmalı ve başka sınıfların sorumluluğunu üstlenmemelidir.//
14
15
16 2 references
public class Cook
17
18 {
19     1 reference
    public bool IsitReady(string input)
20     {
21         return true;
22     }
23 }
24 0 references
public class serve
25
26 {
27     public Cook _cook = new Cook();
28
29     0 references
    public bool ServeforNew(string value, string input)
30     {
31         if (!_cook.IsitReady(input))
32         {
33             return false;
34         }
35         else
36         {
37             return true;
38         }
39     }
40 }
```

2. Bir programın, uygulamanın geliştirmeye açık ancak değiştirmeye kapalı olması özelliğidir. Örneğin print işlemini yapan bir interface tanımladık. Bu şekilde değişikliğe gerek olmadan yeni gelen yazıcılar print özelliğini interface ile kazanmış oldu.



```
42 4 references
public interface Iprint
43
44 {
45     2 references
    string Print();
46 }
47 0 references
public class CanonPrint : Iprint
48
49 {
50     2 references
    public string Print()
51     {
52         throw new NotImplementedException();
53     }
54 }
55 0 references
public class HPPrint : Iprint
56
57 {
58     2 references
    public string Print()
59     {
60         throw new NotImplementedException();
61     }
62 }
63
64 public class Print
65
66 {
67     private Iprint _print;
68
69     0 references
    public void Printwith(Iprint _print)
70     {
71         //yazdırma işlemi
72     }
73 }
```

3.Sırf işlemler birbirlerine benzedikleri için aynı çatı altında toplanmaz. Örneğin aşağıdaki örnekte müşteriler için IBAN’a gerek vardır fakat öğrenciler için IBAN eklenmesi gerekmez.

```
0 references
public class CustomerManager:AddIBAN,AddName
{
}
0 references
public class StudentManager:AddName
{
}

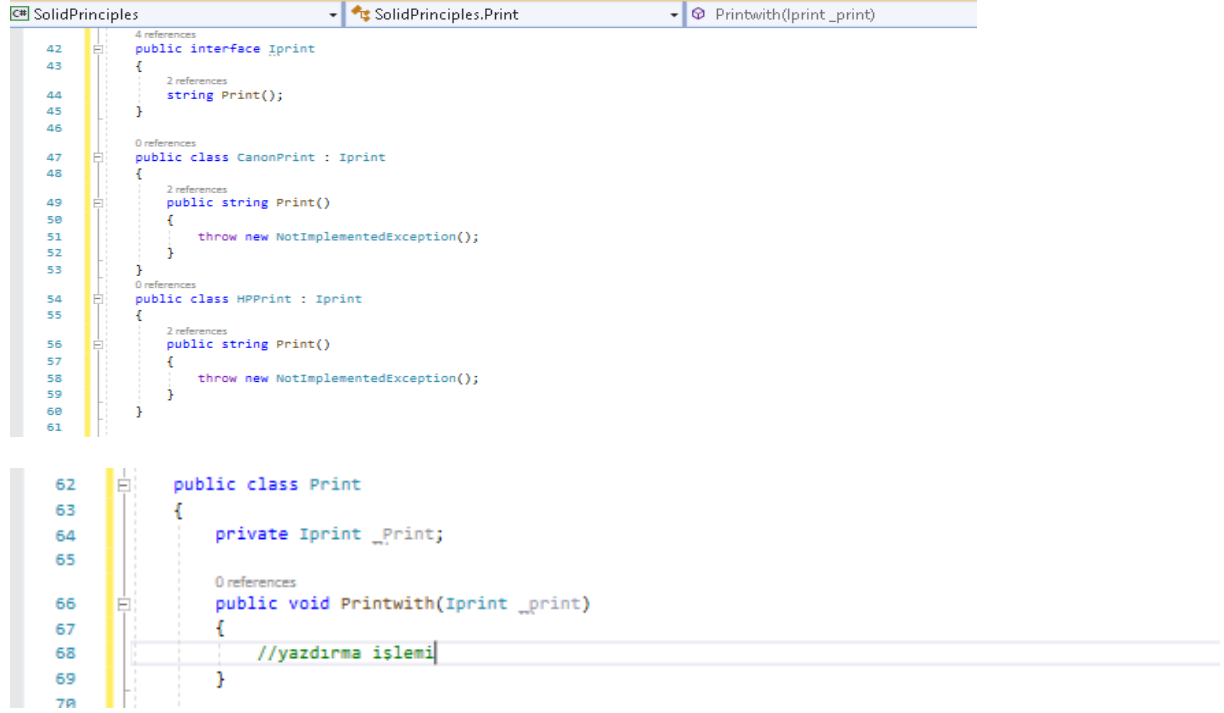
1 reference
public interface AddIBAN
{
}
2 references
public interface AddName
{
}
```

4.Sınıflara gerekli method ve arayüzlerin tanımlanması gerekmektedir. Gereksiz eklentilerden kaçınılmalıdır. Bunun için bir önceki örnek de verilebilir. Her sınıf için gerekli işi yapan arayüz tanımlandı.

```
0 references
public class CustomerManager:AddIBAN,AddName
{
}
0 references
public class StudentManager:AddName
{
}

1 reference
public interface AddIBAN
{
}
2 references
public interface AddName
{
}
```

5. Alt sınıflarda yapılan değişiklikler üst sınıfları etkilememelidir. Yani birbirlerine bağımlılıkları olmamalıdır. Yazdırma örneği bunun için uygundur. Yazdırmayı bir sınıf üzerinde yapıyoruz fakat bağımlılıkları bir interface üzerindedir. Yani üst sınıf(class Print) bu işlemde yazıcılara bağlı değildir.



```
42 public interface Iprint
43 {
44     2 references
45     string Print();
46 }
47
48 0 references
49 public class CanonPrint : Iprint
50 {
51     2 references
52     public string Print()
53     {
54         throw new NotImplementedException();
55     }
56 }
57
58 0 references
59 public class HPPrint : Iprint
60 {
61     2 references
62     public string Print()
63     {
64         throw new NotImplementedException();
65     }
66 }
67
68 public class Print
69 {
70     private Iprint _print;
71
72     0 references
73     public void Printwith(Iprint _print)
74     {
75         //yazdırma işlemi
76     }
77 }
```

.NET Core 3.1'deki yenilikler

.NET Core 3.1 ile ilgili en önemli özellik, uzun vadeli bir destek (LTS) sürümüdür.

Uzun Süreli Destek (LTS) nedir?

Microsoft bunu üç yıl içinde destekliyor, böylece uygulamalarımızı .NET Core 3.1'e taşıyabiliriz. Bu öneriyi aşağıdaki analizlerle inceleyebiliriz.

Sürüm notu

.NET Core 3.0 3 Mart 2020'de kullanım ömrü dolmuş.

.NET Core 2.2 Kullanım ömrü 23 Aralık 2019.

.NET Core 2.1 21 Ağustos 2021'de kullanım ömrü dolmuş.

MacOS uygulaması

AppHost ayarı varsayılan olarak devre dışıdır. AppHost ayarı etkinleştirildiğinde, .NET Core oluşturduğunuzda veya yayınladığınızda yerel bir Mach-O yürütülebilir dosyası oluşturur.

Uygulamanız, dotnet run komutuyla kaynak kodundan çalıştırıldığında veya Mach-O yürütülebilir dosyasını doğrudan başlatarak appHost bağlamında çalışır. kullanıcının çalışma zamanına bağlı bir uygulamayı başlatabilmesinin tek yolu dotnet <dosyaadı.dll> 'dir.

Windows Formları

Visual Studio Designer Toolbox'ta bir süredir kullanılmayan Windows Forms'da. Bunlar, .NET Framework 2.0'daki yeni denetimlerle değiştirildi. Bunlar .NET Core 3.1 için Masaüstü SDK'sından kaldırılmıştır. Araç kutusunda tasarımcılar değiştirildi. Bunlardan biri DataGridView. Yani bu öneri uygulamamızı güncellemektir.

C ++ / CLI

C ++ / CLI ("yönetilen C ++" olarak da bilinir) projeleri oluşturmak için destek eklendi. Bu iş yükü Visual Studio'ya iki şablon ekler:

- CLR Sınıf Kütüphanesi (.NET Core)
- CLR Boş Proje (.NET Core)

Hafta içinde katıldığım etkinlik;

1.Damla Alkan anlatımı ile Yapay Zeka ; Azure Cognitive Search ile dökümanlarımızı Azure Bulut Storage'e yükleyerek bu dökümanlarla görüntü işleme,metin analizi yapabilmeyi deneyimledik. Bulutta var olan yapa zeka hizmetlerini kullandık. Sonrasında makine öğreniminden bahsettik ve eğitim boyunca AI(Yapay Zeka),ML(Makine öğrenimi),DL(Derin öğrenme) kavramlarının üzerinde durduk.

Takip edilebilecek yazılımcılar;

- 1.Engin Demiroğ
- 2.Sadık Turan