**Solid Principles**

Solid prensibinde amaç, yazılımda sürdürülebilirliği sağlamaktır.

1. Single Responsibility Principle (SRP)
2. Open - Close Principle (OCP)
3. Liskov Substition Principle (LSP)
4. Interface Segregation Principle (ISP)
5. Dependency Inversion Principle (DIP)

Amaç:

* Geliştirdiğimiz yazılımın gelecekte gereksinimlere kolayca adapte olması,
* Yeni özellikleri kodda bir değişikliğe gerek kalmadan kolayca ekleyebileceğimiz
* Yeni gereksinimlere karşın kodun üzerinde en az değişimi sağlaması,
* Kod üzerinde sürekli düzeltme hatta yeniden yazma gibi sorunların yol açtığı zaman kaybını da minimuma indirmektir.

**Single Responsibility Principle (SRP)**

Bir sınıf veya bir fonksiyon sadece bir işi yapmalıdır. Bir fonksiyon birden fazla amaça hitap edemez.

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturulduYanlış.

Tek fonksiyonda iki ayrı işlem yapmamalıyız.

Ayrıca iki araba classı olduğunu düşünelim. Ortak parametreleri, fonksiyonları ortak base classa taşımak gerekir.

**Doğru olmayan bir kod:**

**metin, ekran görüntüsü, yazı tipi, doküman, belge içeren bir resim

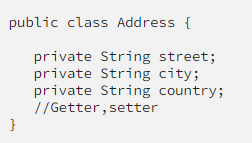
Açıklama otomatik olarak oluşturuldu**

**Doğru olan kodlar:**

metin, yazı tipi, ekran görüntüsü, beyaz içeren bir resim

Açıklama otomatik olarak oluşturuldu

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*



**Open - Close Principle (OCP)**

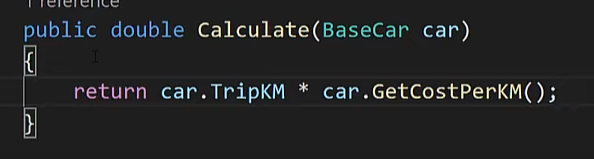
Geliştirmeye açık, değiştirmeye kapalı olma durumu.

metin, ekran görüntüsü, yazılım, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Buraya yeni bir araba eklersem burdaki kodu güncellemem gerekecek bu durumda bu istemeyeceğimiz bir olaydır.

Bunun yerine BaseCar classını abstract tanımlarız ve bir abstract metotla kalıtım alan classlara override etmesini sağlarız.



Kısacası dinamiklik diye özetleyebiliriz. Var olan classı değiştirmeyiz, yeni ekleneni güncelleriz.

**Alan hesaplama örneği:**

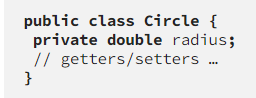
metin, yazı tipi, ekran görüntüsü, beyaz içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin, ekran görüntüsü, yazı tipi, çizgi içeren bir resim

Açıklama otomatik olarak oluşturuldu

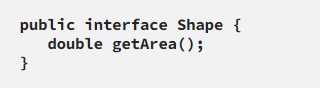
Daire alan hesabı eklememiz gerekti…



metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Burada ortalık karışmaya başladı. Üçgen de eklemeye kalkarsak burayı tekrar değiştirmemiz gerekecek.



Bunu bir interface ekleyerek her şeklin kendi içinde hesaplama fonksiyonunu yazmasını sağlayabiliriz.

metin, yazı tipi, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin, yazı tipi, ekran görüntüsü, cebir içeren bir resim

Açıklama otomatik olarak oluşturuldu

Bu şekilde kod karmaşıklığından kurtulmuş oluyoruz.

**Liskov Substition Principle (LSP)**

Kodlarımızda herhangi bir değişiklik yapmaya gerek duymadan alt sınıfları, türedikleri(üst) sınıfların yerine kullanabilmeliyiz.

Bir class kalıtım aldığı class gibi davranamıyorsa burda bir problem vardır.

BMW SMS göndermek istemiyorsa fakat Mercedes göndermek istiyorsa (ki bu fonksiyon BaseCar içerisinde) bu fonksiyonları **interface** kullanarak ayırmalıyız.

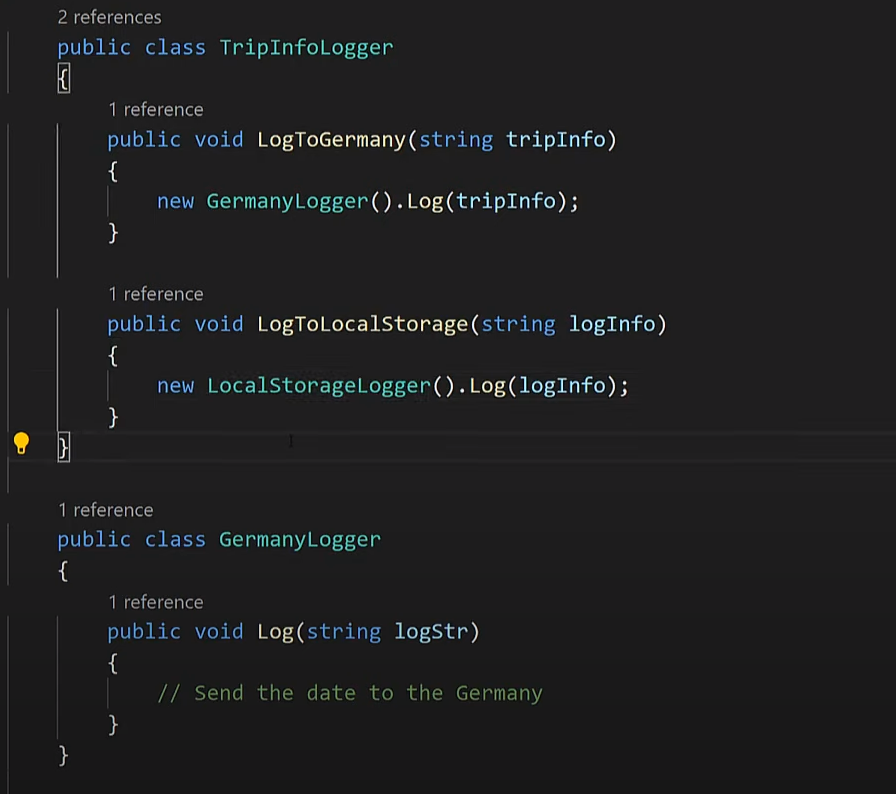
**Interface Segregation Principle (ISP)**

Sorumlulukların hepsini tek bir arayüze toplamak yerine daha özelleştirilmiş birden fazla arayüz oluşturmalıyız.

Bir class içerisinde tanımlamanız gereken bir fonksiyonu diğer classlarda kullanmayacaksak interface içine eklememeliyiz. Yoksa diğer classlar problem yaratacaktır. Bunun için yeni bir interface açmalıyız.

**Dependency Inversion Principle (DIP)**

Daha üst seviyedeki bir class, alt seviyedeki bir classa bağımlı olmamalı.



Her seferinde hata mesajını bir yere göndermek için fonksiyon yazmıyor olmamız gerekiyor.