Single Responsibility Principle ( Tek Sorumluluk Prensibi )

* Bu prensip her bir nesnesin sadece tek bir görev yapmasını isteyen bir prensibtir. Tabii ki bu şu anlama gelmiyor her bir nesne sadece tek bir metoda sahip olmalı demek değildir.
* Örneğin, garsonu ele alalım garson sadece masa toplamaz garsonun garson olarak birden fazla işlevi vardır. Garson, masayı toplar, siparişi alır, siparişi aşçıya veya pişirme bölümüne iletir, Müşteriyi karşılar gibi bir çok görevi vardır. Ama garson gidip yemek yapmaz. Bu prensibimiz de bunu savunuyor garson sadece garson görevlerini yapsın diyor.

Örnekler

* Öğrenci

Öğrenci , ders seçimi yapar, derslere girer, ödevlerini yapar, bunlar öğrencinin görevleridir.

Öğrenci gidip ders anlatmaz veya not girişi yapmaz. Bunları yaparsa SRP e uymayan bir proje ortaya çıkar.

Yazılımcı

Yazılımcı, kod yazar, test yapar,vb.

Yazılımcı eğer çay yapıyorsa bu SRP’e uymaz.

Open-Closed Principle ( Açık-Kapalılık Prensibi )

* Bu presinsib projemizin gelişime açık fakat değişime kapalı olmasını ister. Bunun anlamı projeye tabii ki yenilik eklemek istediğimiz zaman eklebiliriz ama bu eklediğimiz yeniliğin bizim daha önce olan projedeki kısımları değişmemesi gerekiyor.

Örnekler

* Üyelik sistemi olan bir satış sitemiz olduğunu düşünelim. Bu sitede Standart, Gold ve Silver üyelik çeşitlerimiz olsun. Bunu öyle bi kodlayalımki gelişime kapalı olsun örneğin yeni bir üyelik eklemek istiyoruz buda Platinium olsun. Bu üyeliği eklerken tüm sistemin değişmesi gereksin işte bu bizim istemediğimiz bir durum bu yüzden.

Projemizin yapısını öyle kurmalıyız ki yeni bir üyelik seviyesi eklendiğinde bunun için bununla bağlantılı tüm kodları değiştirmemize gerek kalmasın.

Liskov Substitution Principle (Liskov Yerdeğiştirme Prenbisi)

* Bu prensib kodumuzda bir değişiklik yapmadan derived classları base class yerine kullanılabilir olmasını istiyor. Tabii ki derived class base classtan farklı metodlara sahip olabilir fakat base classtan aldığı metodları değiştirmemeli.

Örnek

* Mesela garson clasımızdan türetilmiş bir BaşGarson classı olsun. Başgarson classı garson sınıfındaki her görevi yerine getirebiliyor ve üstüne kendi classına özgü bazı görevleride var mesela GarsonlarıKontrolEt gibi metodu olsun. Biz istediğimiz zaman garson yerine Başgarson derived classını kullanarak garsonun görevlerinide yerine getirebiliriz.

İnterface Segregation Principle ( Arayüzlerin Ayrımı Prensibi )

* Bu presinsib implement alınan interfaceteki her bir metodun veya nesnenin implement alınan classta kullanılmasını istiyor. Eğer İmplement alınan interfaceteki tüm metodlar kullanılmıyor ise bunun ayrılıp özel olarak kullanıcak olan nesneye gore yazılması gerekiyor.

Örnek

* Geometrik şekillerle alakalı alan hesaplayan bir interfacesimiz var. Bu interface içinde Kenar Uzunlukları, Yükseklik , Pi, Yarı çap gibi tüm geometrik şekillerde kullanacağımız değişkenlerimiz var. ve Kare şeklinde nesnemize bu interface implement ettiğimiz zaman karede sadece tek bir kenar uzunluğundan hesap yapımı yeterli olacağından diğer tüm değişkenler kare için boşuna tanımlanmış olucak. İşte biz bunu istemediğimiz için ortak olan değişkenleri ortak bir implemente atıp ortak olmayanları ya özel olarak Şekil classlarımızda yada farklı interfacelere yazmamız gerekiyor böylece gereksiz değişken veya metod tanımlaması yapılmamış olur.

Dependency İnversion Principle ( Bağlılığı Tersine Çevirme Prensibi )

* Bu prensib Genel Ana Classtaki parçaların herhangi birinde değişiklik yapılacağı zaman gidip sadece küçük parçalarda değişiklik yapılıp sisteme uyarlanılması için Küçük Parçalar büyük classlara bağlı değilde Genel Classın küçük parçaya bağlı olduğu bir sistem yapılır

Örnek

* Apartmanı ele alalım. Apartmandaki herhangi bir odada bir sıkıntı olduğu zaman direk apartmanı değiştirmiyoruz. Hangi odada problem varsa odada değişiklik yapıyoruz ve problemi çözmüş oluyoruz.
* Çanta classımız olsun. Bu çanta clasımızın fermuarı var. Fermuar bozulduğu zaman fermuarı değiştiriyoruz ve problem ortadan kalkmış oluyor.