Single Responsibility Principle (SRP) – Tek Sorumluluk Prensibi-1

Merhaba sevgili dostlarım! Söz verdiğim gibi prensipleri incelemeye başlıyoruz. İlk ele alacağımız prensip ise başlıktan anladığınız üzere “Single Responsibility Principle”. Öncelikle altını çizmek istediğim bir durum var. Bu prensipten başlamamın sebebi, en önemli olması değil, SOLID kısaltmasının ilk harfi olmasıdır yanlış anlaşılmasın.

İlk olarak bu prensibin anahtar cümlesini belirtelim: “Bir nesneyi değiştirmek için tek bir neden olmalıdır”. Elbette bu cümleyi biraz daha açarak örneklerle inceleyeceğiz. Ancak ondan önce, anahtar cümlemizi tersine çevirerek bu prensibe uymayan nesneyi nasıl teşhis edeceğimizi bulalım: “farklı amaçlara yönelik birçok metodu içeren bir nesnem var”. İşte size teşhisi kolaylaştıracak bir cümle.

Dostlarım, bir şeyi hatırlatmama izin verin. Burada anlatmakta olduğum prensibe uymamak sizin iyi bir uygulama yazmadığınızı kesinlikle göstermez! En iyi uygulama çalışan uygulamadır velhasıl. Bu prensiplerin en büyük amacı, kodunuzun esnekliğini sağlamaktır. Yani sürekli büyümesi muhtemel bir proje yazıyorsanız, uzun vadede kod bakımı ve geliştirilmesi kolay bir mimari uygulamanız sizin de işlerinizi kolaylaştıracaktır.

Konumuza dönelim. Kısaca SRP olarak adlandırdığımız bu prensip, her nesnenin yalnızca bir işten sorumlu olması gerektiğini söylüyor. Şimdi gözünüzün önüne bir restoranın mutfağını getirin lütfen. Bu mutfaktaki çalışanlardan ton balıklı salata hazırlanması istenmiş. Uzun bir tezgah üzerinde yan yana çalışan insanlar var. İlk sıradaki, domatesleri doğruyor ardından diğeri salatalığı dilimliyor. Üçüncü kişi yeşillikleri yıkayıp güzelce dizdikten sonra dördüncü kişi turşuyu hazırlıyor. Beşinci kişi ton balığını salataya ekliyor ve altıncısı ise hepsini karıştırıp tamamlanan salatayı garsona iletiyor.

İşte buradaki aşçıların her birinin yalnızca bir sorumluluğu var değil mi? Şimdi anahtar cümlemize tekrar bakalım: “Bir nesneyi değiştirmek için tek bir neden olmalıdır”. Domateslerin doğranma şeklini değiştirmek istediğinizde bunu kaç kişiye söyleyeceksiniz? Aha! İşte budur!

Bir yazılımcının görevi problemleri çözmektir. Bu problemler zaman zaman çok karmaşık olabilir. İşte bu noktada SRP yaklaşımı ile büyük problemi küçük parçalara ayırabilirsiniz. Böylece bu mini problemlere odaklanarak büyük problemi çok daha rahat çözersiniz.

Böylece geliştirdiğiniz uygulama tıpkı bir saatin çalışma mekanizması gibi olacaktır. Her bir parçanın tek bir sorumluluğu bulunacaktır. Fakat elbette kod yazarken bu sorumlulukları ayırmak çok da kolay değildir. Bu nedenle, SRP’ye uygun *yazılmamış* bir nesneyi ele alarak başlayalım. Yapacağımız bu işleme yeniden düzenleme (refactoring) dendiğini de belirtelim.

Madem bir restoran örneği ile başladık, benzer bir örnekle devam edelim. Bir yemek yapma simülatörü geliştirdiğinizi düşünelim. Oyuncu, yemeği hazırlayacak, pişirecek ve sunacak. Hadi başlıyoruz:

public class Pizza

{

public string Ad { get; set; }

public string Boyut { get; set; }

public double Fiyat { get; set; }

public void MalzemeEkle(string malzeme)

{

//malzeme ekleme işlemleri

}

public List<string> MalzemeleriGoster()

{

//malzeme göstermek için gereken işlemler

}

public void MalzemeyiCikar(string malzeme)

{

//listeden malzeme çıkarmak için yapılan işlemler

}

public void Pisir(int sure)

{

//Pişirme işlemleri...

}

public void HamuruAc(int boyut)

{

//Hamuru açmak için gereken kodlar...

}

}

Dostlarım, bu sınıf şu anda SRP prensibine uymuyor. Bakalım bu “Pizza” sınıfı hangi işlemleri yerine getiriyor?

1. Pizza nesnesinin belirli değerlerini tutuyor.
2. Malzeme listesi ile ilgili işlemler yapıyor.
3. Pizzanın kendisi ile ilgili işlemler yapıyor.

Bu sınıf içerisine yeni bir metot eklemek ya da var olan metodu güncellemek, kısaca değişiklik yapmak için üç farklı sebebiniz var. Ne demiştik, her sınıfın yalnızca bir sorumluluğu olmalıdır. Haliyle, bu üç sorumluluğu da başka tiplere ayırmalıyım.

Hadi o zaman!

public class Pizza

{

public string Ad { get; set; }

public string Boyut { get; set; }

public double Fiyat { get; set; }

}

public class MalzemeIslemleri

{

public void MalzemeEkle(string malzeme)

{

//malzeme ekleme işlemleri

}

public void MalzemeyiCikar(string malzeme)

{

//listeden malzeme çıkarmak için yapılan işlemler

}

public List<string> MalzemeleriGoster()

{

//malzeme göstermek için gereken işlemler

}

}

public class PizzaIslemleri

{

public void Pisir(int sure)

{

//Pişirme işlemleri...

}

public void HamuruAc(int boyut)

{

//Hamuru açmak için gereken kodlar...

}

public MalzemeIslemleri Malzemeler { get; set; }

}

İşte! Şimdi her sınıf yalnızca tek bir işten sorumlu! PizzaIslemleri sınıfı, Pişirme, Hamur açma ve pizza malzemeleri ile ilgili işlemlerden sorumlu. Fakat malzeme işlemleri, başka bir değişim sebebi olduğundan malzeme ekleme, çıkarma ve listeyi gösterme işlerinin sorumluluğu MalzemeIslemleri sınıfına ait. Pizza sınıfı ise diğer sınıflardan tamamen bağımsız olarak sadece nesnenin özelliklerini barındırıyor!

Şimdi düşünün lütfen. Bu sınıfların bakımı ve geliştirilmesi daha kolay değil mi? Ama en önemlisi anlaşılması (okunması) daha kolay.

Tamam. Bu ilk SRP makalesinde örneğimizi basit tutalım. Böylece sadece SRP’nin temel mantığını kavramış olalım. Ama durun! SRP’nin kendisi kolay olsa da uygulanması hiç de kolay değildir. İşte bu nedenle bir sonraki SRP makalemizde biraz daha karmaşık bir uygulamayı ele almayı planlıyorum.

Şimdilik hoşça kalın.