11.02.2022

PATİKA – INNOVA JAVA SPRİNG
ODEV – 4

Bekir Gürkan Güldaş

İçindekiler

1. SOLID Prensipleri

S — Single responsibility principle
● Bir sınıfın veya fonksiyonun yapması gereken yalnızca bir işi olması gerekir.
O — Open closed principle
● Bir sınıf veya fonksiyonun halihazırda var olan özellikleri korumalı ve değişikliğe izin vermemelidir. Ancak yeni özellikler kazanabiliyor olmalıdır.
L — Liskov substitution principle
● Kodlarda herhangi bir değişiklik yapmaya gerek duymadan alt sınıflar, türedikleri üst sınıflar yerine kullanılabilmelidir.
I — Interface segregation principle
● Sorumlulukların hepsini tek bir arayüze toplamak yerine daha özelleştirilmiş birden fazla arayüz oluşturulmalıdır.
D — Dependency Inversion Principle
● Sınıflar arası bağımlılıklar olabildiğince az olmalıdır özellikle üst seviye sınıflar alt seviye sınıflara bağımlı olmamalıdır.

1.1. Single Responsibility

Single responsibility prensibi bir sınıf veya foksiyon geliştirileceği zaman iyi tanımlanmış tek bir sorumluluğu olması gerektirdiğini anlatmaktadır. Bir sınıfın yalnızca bir amaçının olması ve yalnızca bu amaç uğruna değiştirilebilir olması gerekir.

Bır sınıfın sorumluluğunun en aza indirgenmesi o sınıfın değişime daha kolay adapte olmasını sağlamaktadır.

- Bir sorumluluğu olan bir sınıfta çok daha az sayıda test-case olacaktır.
- Tek bir sınıfta daha tek bir sorumluluğunun olması daha az bağımlılık sağlayacaktır.
- Daha az sorumluluk daha yalın veya küçük yapılar ulaşmasını sağlar. Bu ise okunurluğunu artırmaktadır.

1.2. Open Closed

Open-Closed prensibi sınıf için yeni davranışlar eklenebilmesini sağlar. Gereksinimler değiştiğinde, yeni gereksinimlerin karşılanabilmesi için bir sınıfa yeni veya farklı davranışlar eklenebilir olmalıdır. Ancak temel özelliklerinin değişimi ise mümkün olmamalıdır.

Bu sebeple open-closed prensibi bir sınıfa yeni gelecek özellikler için varolan kodu değiştirmeden, varolan yapıyı bozmadan esnek bir geliştirme modeli uygulayarak, önü açık ve gelecekten gereksinimlere kolayca adapte olabilen bir model uygulaması gerektiğini anlatmaktadır.

1.3. Liskov Substitution

Alt seviye sınıflardan oluşan nesnelerin, üst sınıfın nesneleri ile yer değiştirdikleri zaman, aynı davranışı sergilemesi gerekmektedir. Türetilen sınıflar, türeyen sınıfların tüm özelliklerini kullanabilmelidir ve kendine ait yeni özellikler barındırabilmelidir. (Polymorphism)

1.4. Interface Segregation

Interface segregation prensibine göre nesneler asla ihtiyacı olmayan metotları içeren arayüzleri implement etmeye zorlanmamalıdır. Birden fazla amaç için yalnızca bir arayüz mevcut ise ilgili sınıfa gerektiğinden fazla metot ya da özellik ekleniyor demektir. Bu sebeple tek bir arayüz yerine kullanımlarına göre parçalanmış birden fazla arayüz ile işlemler yürütülmelidir. Yani her farklı sorumluluğun kendine özgü bir arayüzü olması gerekmektedir.

1.4. Dependency Inversion

Dependency inversion prensibine göre bir sınıfın veya metodun , onu kullanan diğer sınıflara karşı olan bağımlılığını en aza indirgenmelidir. Bir alt sınıfta yapılan değişiklikler üst sınıfları etkilememelidir. Yüksek seviye sınıflarda bir davranış değiştiğinde, alt seviye davranışların bu değişime uyum sağlaması gerekir. Ancak, düşük seviye sınıflarda bir davranış değiştiğinde, üst seviye sınıfların davranışında bir bozulma meydana gelmemelidir.

https://github.com/gurkanguldas/Innova.SolidPrinciple