metin içeren bir resim

Açıklama otomatik olarak oluşturulduRef

Yukarıdaki örnekte Kare() metoduna “a” argümanını gönderip aktardığı değerle işlem yaptırdık. Ve daha sonra “a” değişkenini ekrana yazdırdık. Ancak metodu çağırdıktan sonra da öncede sonuç değişmedi. Çünkü biz “a” argümanının yalnızca değerini gönderdik. Ama referansını aktarmadık. Bu yüzden yaptığımız işlem yalnızca “i” parametresini ilgilendiriyor. Eğer argümanın gönderdiğimiz metotta kendi referansı ile işlem görmesini istiyorsak imdadımıza ref anahtar sözcüğü yetişiyor. Aynı işlemi ref olmadan da yapabiliriz fakat işler daha karmaşık hale geldiğinde zorluk yaşamamak için ref kullanımını daha mantılı olacaktır. ref parametresi kullanımında önemli bir nokta var. ref olarak aktarılacak bir argümana çağrıdan önce değer atanmalıdır. ref kullanarak bir argümana ilk değer atanması mümkün değildir. Şimdi yukarıdaki işlemin biraz daha karmaşığını ref ile yapalım.

Out

out anahtar sözcüğü iki veya daha fazla geriye döndürmek istediğimizde kullandığımız bir anahtar sözcüktür. ref parametresi de aynı işlemi gerçekleştirebilir. Ancak iki önemli nokta var;

a) out parametresi ilk değer atamasına gerek duymaz. Bu iyi bir özellik, yani sadece geriye değer döndürmek için kullanılır.

b) Metot sonlanmadan out parametresine değer atamak zorundayız, çünkü ilk değer vermediğimiz için herhangi bir değere sahip değil.

Sonuç olarak birden fazla değer döndürme ihtiyacı duyduğumuz zaman out parametresi işimize yarayacaktır. Örnek olarak bir out parametresi kullanımına bakalım.

Burada “b” argümanı ile değeri gönderip, sayi parametresi değerimizi döndürüyoruz. Eğer ref parametresini kullanmış olsaydık “b” argümanına ilk değer atamak zorunda kalacaktık. Konuyu kısaca özetlersek argümanlarımız ile daha esnek işlemler yapabilmek, birden fazla değer döndürebilmek gibi işlemleri ref ve out ile kolayca yapabiliyoruz. Umarım faydalı bir yazı olmuştur.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

OOP

Yazılım dünyasının neredeyse kalıcı temellerinin atılmaya başlandığı geç 1960 larda çıkan bu teknik yazılım dünyasında o zamanda adeta bir devrim yaratmıştır. Nesne yönelimi 4 bacaklı bir iskelet üzerinde şekillendirilebilir. Bu bacaklar, Encapsulation, Inheritance, Polymorphism ve Abstraction bacaklarıdır. Bu Abstraction yani soyutlama sınıf tanımlarında yalnızca gerekli özelliklerin üzerine odaklanılıp buradan belirli yapılar elde etme işidir. Ana amacı gereksiz karmaşıklığı elimine etmek böylece gereksiz detaylardan kullanıcıyı kurtarmaktır. Örneğin çok meşhur olan kahve makinesi örneğini verelim. Kahve makinesinin kendi içerisindeki işlemlerinin varlığı, girdileri bir ürüne dönüştürme süreci ve bu süreçteki kararlarının listesi yalnızca kahve içmek isteyen bir insan için oldukça gereksiz detaylardır. Sonuç odaklı düşünen bir kahve sever sadece kahveyle ilgileneceğinden ☕ onun için gerekli soyutlama işlemlerinin yapılması gayet kritik önem taşımaktadır. Yani kullanıcıya sunacağın basit bir arayüz ve o kahve makinesinin talep edilen kahveyi üretmesi sonucu dışındakilerin pek kıymeti yoktur.

Biraz da encapsulation üzerinde durmakta fayda var. Soyutlamada dışarıdan sonuç odaklı bakılan bir işlem için gereksiz detayların gizlenmesi demiştik. Kapsüle alma yani encapsulation işleminde ise bunu içten süreci yorumlama işlemi üzerinden gereksiz detayları elimine ederek kritik işlemi korumaya almak diyebiliriz. Yani şöyle ki belirli erişim atamalarıyla verilerin erişilebilirlikleri üzerinden ana işlem korumalı hale getirilecektir. Bunu kalemin iç aksamını yerine göre korumak amaçlı onu bir plastik dış gövde içerisine alma işlemi olarak görebiliriz. Böylece veriler ya gizlenir ya erişime açılır ve kullanım üzerinden şekillendirilir.

Sonraki odak noktamız ise Inheritance dır. Inheritance kelime kökeni olarak inherit etme yani atadan elde etme/miras alma anlamına gelir. Mevcut bir nesne üzerinden düşünürsek eğer diğer bir nesneye verilerin aktarılması ile alakalı olduğunu söyleyebiliriz. Mirasın alındığı sınıf Super sınıf olarak adlandırılır, alan ise mirasçı/ara sınıf olarak adlandırılmaktadır. Tekrar kullanım özelliğini ana amaç olarak düşünen Inheritance da tekrar kullanımların varlığıyla birlikte verimliliğin arttırıldığını kolaylıkla söyleyebiliriz. Extends yapısıyla inherit işlemi tanımlanır, ara class yani child class ana yapıdan yani süper class dan gerekli mirasını böylece alır.

Son bacak olan Polymorphism üzerinde de durup bu kısmı bitirelim. Polymorphism, süper class üzerinden miras alınmış birtakım özelliklerin child class üzerinde değişen talebe bağlı olarak değiştirilmesi özelliğine denir. Neden böyle bir şeye ihtiyaç olunsun yahu zaten ihtiyaç o değilse neden miras alınıyor dediğinizi duyar gibiyim. Örneklemekten zarar gelmez öyle değil mi?

DI ve IOC

Kabaca, kodumuz çalışacağı zaman framework’ün kodumuzu çağırması, çalıştırması ve kontrolü tekrar ele alması sürecine Inversion of Control denir. Biraz daha teknik ifadeleri verecek olursak; bir uygulamanın bir kısmının ya da objelerinin kontrolünün bir framework’e verilmesi olarak tanımlayabiliriz. Bu yapının tanımı genellikle OOP context’i içerisinde kullanılır. Kod içerisinde bir çağrım yaptığımızda framework’ün program akışını ele alması ve kodumuz üzerinde çağırımlar yapmasına imkan tanır. Bunu sağlamak için de framework’ler ek bazı davranışlara sahip abstractionları kullanır. Eğer kendi davranışımızı da eklemek istersek framework’ün ilgili yapısını kendi sınıflarımıza extend etmemiz gerekir.

Dependency Injection aslında IoC’nin implement edilmesini sağlayan bir pattern’dır ve programımızın ihtiyacı olan bağımlılıkları sağlaması amaçlanır. Kısacası kod içerisinde inject ettiğimiz bağımlılıklarımız için bizim buna ihtiyacımız var git bunu getir ve bu sınıf içerisinde inject et diyeceğimiz bir konsepti ifade ediyor.