**DESİGN PATTERN**

Analiz - Tasarım - Develop - Test

Design Patterns, yazılım tasarımında kullanılan genel çözüm şablonlarıdır. Nesne yönelimli programlama ile yakından ilişkilidirler ve OOP prensiplerini uygulamak için rehberlik sağlarlar. Bu desenler, nesneler arasındaki ilişkileri, davranışları ve yapıları tanımlar, böylece yazılımın daha modüler, esnek ve bakımı daha kolay hale gelmesini sağlarlar. Ayrıca, Design Patterns'in kullanımı, yazılım geliştiricilerin problemleri ele alırken daha sistemli ve etkili olmalarına yardımcı olur. Bu desenler, birlikte çalışan nesneleri ve bileşenleri organize etmek ve birlikte çalışabilirliği artırmak için kullanılabilir. Kod değildir.

Zaman ve efor kazandırır. Dünya genelinde 250 den fazla var ancak 23 tanesi standarttır.

**Oluşturucu(Creational) Patterns:** Nesnelerin oluşturulmasında sıklıkla karşılaşılan sorunlara çözüm için;

Singleton Pattern: Nesnenin sadece bir defa oluşturulmasını öngörür.

Factory Pattern: Nesnenin nasıl yaratılacağını kalıtım yoluyla alt sınıflara bırakıp nesne yaratımı için tek ara yüz kullanarak, ara yüzle nesne yaratım işlevlerini temelde birbirinden ayırmaya yarar.

Abstract Factory Pattern: Yapısal olarak birbirine benzeyen ürünlerin ortak bir ara katman üzerinden yönetilebilmesini sağlamaktır. Tek ara yüz kullanarak bir nesne ailesinin farklı platformlarda yaratılmasına olanak sağlar. Diğer bir deyişle, uygulama farklı platformlara taşınabilir.

**Yapısal(Structural) Patterns:** Nesneler arasındaki yapılarda sıklıkla karşılaşılan sorunlara çözüm için;

Adapter Pattern: Sadece bir sınıfa özel olan arayüzleri diğer sınıflarla uyumlu arayüzler haline getirir.

Bridge Pattern: Soyutlamayı (abstraction) uygulamadan (implementation) ayırarak ikisinin birbirinden bağımsız çalışmasını sağlar.

Filter Pattern: Belirli bir veri kümesinden istenen öğeleri seçmek veya dışlamak için kullanılır ve genellikle koleksiyonlarda veya listelerde kullanılır. Örneğin, bir e-ticaret sitesinde ürünleri filtrelemek için kullanılabilir.

Composite Pattern: Birbirilerinden farklı olan bir grup nesnenin sanki tek bir bütün nesneymiş gibi davranmalarını sağlarlar.

Decorator Pattern: Runtime esnasında bir sınıfın işlevlenebilirliğinin genişletilmesini veya başka bir deyişle yeniden dekore edilebilmesini mümkün kılar.

Facade Pattern: Çok geniş boyutlardaki kod parçalarını onlara göre çok daha sadeleştirilmiş arayüzlere indirgenilmesini sağlar.

Flyweight Pattern: Çok sayıda aynı türden nesneler yaratılacağına tek bir nesneden görsel nesneler yaratarak kalabalık bir nesne yapısı kurulmasına olanak sağlar.

Proxy Pattern: Yüksek boyutta dosyalar ve yapım aşaması zorlayıcı olan nesnelerin tek bir arayüz tarafından kullanılmalarını olası kılar.

**Davranışsal(Behavioral) Patterns:** Nesnelerin davranışlarını etkilemede ve yönetmede sıklıkla karşılaşılan sorunlara çözüm için;

Chain of Responsibility Pattern: Servis sağlayan ilgili tüm nesneler birbirleriyle ilişkili hale getirilir. Bir nesne zincirdeki kendinden sonraki nesneyi tanır ve isteğe cevap veremediği durumda, kendinden sonraki nesneye iletir. Bu işlem, zincirde bulunan doğru servis sağlayıcı nesneyi bulana kadar devam eder.

Command Pattern: Kullanılmak istenen nesnenin tanınmadığı durumlarda, komut tasarım kalıbı ile yapılmak istenen işlem bir nesneye dönüştürülerek, alıcı nesne tarafından işlemin yerine getirilmesi amaçlanmaktadır.

Interpreter Pattern: Sıklıkla karşılaşılan belli mantıksal kalıpların bir bütün içerisinde yer almasını sağlamak amacıyla kullanılmaktadır. Dil bilgisi kuralları gibi kalıplar içerisinde yer alan ifadelerin yorumlanması amacıyla kullanılması tercih edilmektedir.

Iterator Pattern: Bir listede yer almakta olan nesnelerin sırasıyla, listenin yapısını ve çalışma tarzının uygulamanın diğer kısımları ile olan bağlantılarını en aza indirmek için uygulamadan soyutlama amaçlı kullanılabilmektedir.

Mediator Pattern: Nesnelerin yönetimi ve aralarındaki iletişimin merkezi bir noktadan sağlanması ve yönetilmesi için kullanılmaktadır.

Memento Pattern: Genellikle geri alma işlemleri gibi durumları yönetmek için kullanılır. Bir nesnenin durumunu kaydedip daha sonra geri yüklemek için kullanılır.

Observer Pattern: Uygulama içerisinde bir nesnede meydana gelen değişikliklerden haberdar olup üzerinde belli metotları çalıştırıp değişlikler yapmak isteyen diğer nesneler bulunabilmektedir. Bu durumda haberdar olmak isteyen nesneler diğer nesne ile ilişkilendirilerek, ilişkili oldukları nesnede meydana gelen değişikliklerden haberdar edilebilmektedirler.

Null Object Pattern: Bir nesne beklenen bir işlevi gerçekleştiremiyorsa, null nesnesi kullanılır ve bu nesne, beklenen davranışı gerçekleştiren ancak hiçbir şey yapmayan bir sınıftır.

Strategy Pattern: Algoritmayı bir nesne içine kapsülleyerek algoritmanın bağımsız olarak değiştirilmesine olanak tanır. Bu desen, birçok algoritmanın birbiriyle değiştirilebilir olduğu durumlarda kullanışlıdır.

State Pattern: Bir nesnenin metotları, sahip olduğu özellikler değiştiği zaman değiştirilebilir. Bu durumda nesnenin sahip olduğu sınıf yapısı değiştirilmiş olacaktır.

Visitor Pattern: Bir sınıf hiyerarşisinde yer alan sınıflar üzerinde değişiklik yapmadan, bu sınıflara yeni metotların eklenmesini kolaylaştırır.