Design Patterns (Tasarım Desenleri) Nedir?

**Tasarım desenleri, sıkça karşılaşılan ve birbirine benzeyen sorunları çözmek için geliştirilmiş esnek kalıplardır.**   
Nesne yönelimli programlamada, **programlama dili**gözetmeksizin; sınıf ve nesneler arasındaki ilişkilerin en iyi şekilde nasıl olmaları gerektiğini açıklayan yöntemlerdir. Kısaca aklın yolu bir diyerek uyguladığımız hazır çözümlerdir.

Yazılım mühendisliğinde tasarım deseni, yazılım tasarımında sık karşılaşılan bir soruna genel tekrarlanabilir bir çözümdür. Tasarım deseni doğrudan koda dönüştürülebilen bir şablon değildir. Birçok farklı durumda kullanılabilecek bir sorunun nasıl çözüleceğine dair bir açıklama veya şablondur.

Tasarım Kalıplarının Kullanım Alanları

Tasarım kalıpları, test edilmiş ve kanıtlanmış geliştirme imkanları sağlayarak geliştirme sürecini hızlandırabilir. Etkili yazılım tasarımı, uygulamanın ilerleyen zamanlarına kadar görünmeyebilecek sorunları dikkate almayı gerektirir. Tasarım kalıplarının yeniden kullanılması, büyük sorunlara neden olabilecek küçük sorunların önlenmesine yardımcı olur ve kalıplara aşina olan kodlayıcılar ve mimarlar için kod okunabilirliğini artırır.

Çoğu zaman, insanlar sadece belirli yazılım tasarım tekniklerinin belirli sorunlara nasıl uygulanacağını anlarlar. Bu tekniklerin çok çeşitli sorunlara uygulanması zordur. Tasarım modelleri, belirli bir soruna bağlı özellikler gerektirmeyen bir biçimde belgelenmiş genel çözümler sunar.

Buna ek olarak, kalıplar geliştiricilerin yazılım etkileşimleri için iyi bilinen, iyi anlaşılan isimler kullanarak iletişim kurmalarını sağlar. Yaygın tasarım desenleri zaman içinde geliştirilebilir. Bu da onları geçici tasarımlardan daha sağlam hale getirir.

**Tasarım desenlerini 3 ana başlıkta değerlendiririz;**

**1. Oluşturucu(Creational) tasarım desenleri**  
Nesnelerin nasl yaratılacağı hakkında öneriler sunar. Bu tasarım kalıpları tamamen sınıf örneklemesi ile ilgilidir. Bu örüntü ayrıca sınıf yaratma örüntülerine ve nesne yaratma örüntülerine ayrılabilir. Sınıf yaratma kalıpları örnekleme sürecinde kalıtımı etkin bir şekilde kullanırken, nesne oluşturma kalıpları işi yapmak için delegasyonu etkin bir şekilde kullanır.

* **Abstract Factory (Soyut Fabrika) :** Birkaç sınıf ailesinin bir örneğini oluşturur.
* **Builder (Kurulum) :** Nesne yapısını temsilinden ayırır.
* **Object Pool :**Artık kullanılmayan nesneleri geri dönüştürerek pahalı kaynaklardan kurtulmak ve kaynakları serbest bırakmak için kullanılır.
* **Factory (Fabrika)**
* **Factory Method (Fabrika Metodu)** :  Birkaç türetilmiş sınıfın örneğini oluşturur.
* **Prototype (Prototip) :** Kopyalanacak veya klonlanacak tamamen başlatılmış bir örnek oluşturur.
* [**Singleton (Tek Nesne)**](https://www.rizagunes.com/singleton-tek-nesne-design-pattern) **:** Yalnızca tek bir örneği var olabilecek bir sınıf.

**2. Yapısal(Structual) tasarım desenleri**  
Sınıflar arasındaki ilişkileri belirleme önerileri sunar, Nasıl türetilmeli, hangi nesneleri içermeli, sınıflar birbirlerini nasıl bağlı olmalı gibi soruların cevaplarını yanıtlar. Bu tasarım kalıpları tamamen Sınıf ve Nesne kompozisyonuyla ilgilidir. Yapısal sınıf yaratma kalıpları, arayüz oluşturmak için kalıtım kullanır. Yapısal nesne kalıpları, yeni işlevsellik elde etmek için nesne oluşturma yollarını tanımlar.

* [**Adapter (Adaptör)**](https://www.rizagunes.com/adapter-adaptor-design-pattern) **:** Farklı sınıfların eşleme arayüzleri.
* **Bridge (Köprü) :** Bir nesnenin arayüzünü uygulamasından ayırır.
* **Composite (Ağaç Yapısı) :** Basit ve kompozit nesnelerin ağaç yapısı.
* **Decorator (Dekorasyon) :** Nesnelere dinamik olarak sorumluluk ekleme.
* **Facade (Ön Yüz) :**Bütün bir alt sistemi temsil eden tek bir sınıf.
* **Flyweight (Sinek Siklet) :**Verimli paylaşım için kullanılan bir örnek.
* **Proxy (Vekil) :**Başka bir nesneyi temsil eden nesne.
* **Private Class Data :**Erişimi kısıtlar.

**3. Davranışsal tasarım desenleri**  
Soruna göre nesneler arası ortak haberleşmeyi en iyi şekilde kurmamıza öneriler sunar. Bu tasarım kalıpları tamamen Class'ın nesne iletişimi ile ilgilidir. Davranışsal örüntüler, nesneler arasındaki iletişim ile en özel olarak ilgili örüntülerdir.

* **Chain of Responsibility (Sorumluluk Zinciri) :** Bir nesne zinciri arasında istek iletmenin bir yolu.
* **Command (Komut) :** Bir komut isteğini nesne olarak kapsülleme.
* **Interpreter (Yorumlayıcı) :** Bir programa dil öğelerini dahil etmenin bir yolu.
* **Iterator (Tekrarlayıcı) :** Bir koleksiyonun öğelerine sırayla erişin.
* **Mediator (Arabulucu) :** Sınıflar arasındaki basitleştirilmiş iletişimi tanımlar.
* **Memento :**Bir nesnenin dahili durumunu yakalama ve geri yükleme.
* **Null Object :**Bir nesnenin varsayılan değeri olarak çalışmak üzere tasarlanmıştır.
* **Observer (Gözlemci) :** Bazı sınıflardaki değişikliği bildirmenin bir yolu.
* **State (Durum) :**Bir nesnenin durumu değiştiğinde davranışını değiştirme.
* **Strategy (Strateji) :** Sınıf içindeki bir algoritmayı içine alır.
* **Template Method (Şablon Metodu) :** Bir algoritmanın tam adımlarını bir alt sınıfa erteleme.
* **Visitor (Ziyaretçi) :** Değişiklik olmadan bir sınıfa yeni bir işlem tanımlar.

**Tasarım desenleri bize ne sağlar?**

Hepimiz bir şey kodlarken gözümüzden kaçan, yada o an hatırlayamadığımız ve ileride bize sorun teşkil edecek hatalar yapabiliriz. İlk başta bunu engeller. Ortak bir dil konuştuğumuz için özellikle takım çalışmalarında büyük kolaylık sağlar. Bir koda baktığınızda "bu adam hangi kafayla bunu yazmış?" sorununu çözer.