Creational Patterns (Yaratımsal Desenler)

Singleton Pattern:

Singleton tasarım deseni, bir sınıfın yalnızca bir örneğini oluşturmasını ve bu örneğe global bir erişim noktası sağlamasını amaçlar. Bu desen, bir uygulama içinde tek bir nesnenin paylaşılmasını ve bu nesneye bir noktadan erişilmesini sağlamak için kullanılır.

Örnek: Veritabanı bağlantı sınıfları, günlük kayıtları tutan sınıflar gibi kaynak yoğun işlemlerde tek bir örneğin paylaşılması isteniyorsa Singleton deseni kullanılabilir.

Factory Pattern:

Factory tasarım deseni, bir sınıfın nesne oluşturma sürecini alt sınıflara devretmek için kullanılır. Ana sınıf, nesne oluşturmakla görevlendirilen bir veya daha fazla alt sınıfa sahiptir ve alt sınıfların hangi nesneyi oluşturacaklarını belirlemelerine olanak tanır.

Örnek: Bir şekil fabrikası düşünelim. Şekil sınıfından türeyen alt sınıflar (Dikdörtgen, Üçgen, vb.) farklı şekilleri oluşturabilirler.

Abstract Factory Pattern:

Abstract Factory tasarım deseni, bir aile içindeki ilişkili veya bağımlı nesneleri oluşturmak için kullanılır. Bu desen, bir nesne grubunun (örneğin, bir UI kitaplığındaki düğme ve metin kutusu gibi) bir arada çalışmasını sağlamak için kullanılır.

Örnek: GUI (Grafiksel Kullanıcı Arayüzü) kitaplıkları düşünelim. Bir tema veya stil sınıfı içinde, o temaya ait düğme, metin kutusu ve diğer arayüz öğelerini oluşturan bir abstract factory kullanılabilir.

Builder Pattern:

Builder tasarım deseni, bir nesnenin karmaşık oluşturulma sürecini adım adım işlemek ve farklı sunumlarını sağlamak için kullanılır. Bu desen, bir nesnenin yapılandırma adımlarını ayrıştırır ve nesneyi adım adım inşa etmek için bir arayüz sunar.

Örnek: Bir bilgisayar nesnesi oluşturmak için bir builder deseni kullanılabilir. Bu bilgisayarın işlemci, bellek, depolama ve diğer özelliklerini adım adım belirleyen bir builder sınıfı olabilir.

Structural Patterns (Yapısal Desenler)

Adapter Pattern:

Adapter tasarım deseni, iki farklı arayüzü birbirine bağlamak için kullanılır. Bu sayede, uyumsuz olan iki sistemi birlikte çalıştırabiliriz. Bu desen, mevcut bir sınıfın arayüzünü değiştirmek veya başka bir sınıfın arayüzünü kullanabilmek için kullanılır. Adapter, iki uyumsuz arabirim arasında bir köprü görevi görür.

Örnek: USB adaptörleri, farklı cihazlar arasında bağlantı kurmamıza yardımcı olur. Bir cihazın konektörü ile diğer cihazın konektörü arasındaki farkı giderir.

Bridge Pattern:

Bridge tasarım deseni, soyutlamayı uygulamayı ve uygulamayı soyutlamadan ayırmayı amaçlar. Bu desen, bir sınıfın hem soyutlanabilir bir arayüzle hem de bu arayüzü uygulayan bir somut sınıfla ilişkilendirilmesini sağlar. Böylece, hem soyutlama hem de uygulama, kendi başlarına değiştirilebilir hale gelir.

Örnek: Grafik arayüzlerde, bir pencerenin çizim araçları gibi soyutlama ile uygulamayı birbirinden ayırmak için kullanılabilir.

Composite Pattern:

Composite tasarım deseni, tekil nesneleri ve bunların bir araya gelerek bir bütün oluşturmasını işleyen bir tasarım desenidir. Bu desen, nesneler arasında hiyerarşik bir yapı oluşturmak için kullanılır. Hem tekil nesneler hem de bu nesnelerin oluşturduğu bileşik nesneler, kullanım açısından birbirine benzer.

Örnek: Bir dosya sistemi düşünün. Hem dosyalar (tekil nesneler) hem de klasörler (bileşik nesneler) aynı hiyerarşik yapı içinde bulunabilir. Bu desen, bu tür yapıları modellendirmek için uygundur.

Decorator Pattern:

Decorator tasarım deseni, bir nesneyi dinamik olarak genişletmeye olanak tanır ve nesne üzerinde değişiklik yapmadan yeni sorumluluklar ekler. Bu desen, bir sınıfın özelliklerini genişletmek ve alt sınıflar oluşturmadan yeni işlevsellik eklemek için kullanılır. Decorator, "sarma" veya "wrapper" olarak düşünülebilir.

Örnek: Bir kahve siparişi alalım. Temel kahve sınıfına (örneğin, Espresso) ekstra malzemeler eklemek istiyorsak (örneğin, süt, şeker), bu ekstraları bir decorator sınıfı aracılığıyla ekleyebiliriz.

Behavioral Patterns (Davranışsal Desenler)

Strategy Pattern:

Strategy tasarım deseni, algoritmayı tanımlayıp birbiriyle değiştirilebilir hale getirir ve istemciyi bir algoritmanın uygulanmasından bağımsız hale getirir. Bu desen, bir dizi algoritma tanımlar, her birini kapsülleştirir ve bu algoritmaları birbirinin yerine kullanabilir kılar.

Örnek: Ödeme işlemlerini ele alalım. Bir alışveriş sepeti uygulamasında ödeme yöntemlerini temsil eden bir interface oluşturabiliriz. Ardından, bu arayüzü uygulayan farklı ödeme stratejilerini (kredi kartı, PayPal, vb.) içeren sınıfları oluşturabiliriz.

State Pattern:

State tasarım deseni, bir nesnenin durumunu değiştirmek ve bu duruma bağlı olarak davranışını değiştirmek için kullanılır. Bu desen, bir nesnenin iç durumunu temsil eden bir dizi sınıf oluşturur ve bu durumu değiştiren bir "Context" sınıfını içerir.

Örnek: Bir otomobil durumu düşünelim. Otomobilin durumu (örneğin, hareket halinde, dururken, vb.) otomobilin davranışını etkiler. State deseni, bu durumları temsil eden sınıfları kullanarak otomobilin davranışını değiştirmemize olanak tanır.

Oberserver Pattern:

Observer Pattern (Gözlemci Tasarım Deseni), bir nesnenin durumu değiştikçe bağımlı nesnelerin otomatik olarak bilgilendirilmesini sağlayan bir tasarım desenidir. Bu desen, bir nesnenin durumu değiştikçe bağımlı nesnelerin otomatik olarak güncellenmesi gereken durumları ele alır.

Örnek; Diyelim ki bir hava durumu uygulaması yapıyorsunuz. Hava durumu servisiniz sürekli olarak güncellenen bir hava durumu verisine sahip. Kullanıcı arayüzü bu veriyi görüntülüyor ve her güncelleme olduğunda kullanıcıya bildirim göstermek istiyorsunuz.

Iterator Pattern:

Iterator Pattern, bir koleksiyonun içindeki elemanlara sırayla erişimi standartlaştıran bir tasarım desenidir. Bu desen, bir koleksiyonun içinde dolaşmak ve koleksiyon elemanlarına erişmek için bir arabirim sağlar. Bu sayede koleksiyonun iç yapısı değişse bile, kullanıcı kodu değişmeden kalabilir.

Örnek; Diyelim ki bir kitap kütüphanesi uygulaması yapıyorsunuz. Kütüphanenizdeki kitapları sırayla dolaşmak istiyorsunuz.