Konu: Design Patterns

Not: Konu ile ilgili örneklere “https://github.com/BerkeCanGoktas/Staj.git” adresindeki repositoryden erişilebilir.

0-Design Pattern Nedir?

Design patternlar özellikle nesneye yönelik programlamada, sık karşılaşılan durumların çözüm tarifleridir. Belirli bir dile bağlı değillerdir ve bu fikirler hemen her dilde uygulanabilir. Bu fikirleri bilmek her seferinde sıfırdan kod tasarlamakla uğraşmamak ve ortak bir dil oluşturmak açısından önemlidir. Creational, structural ve behavioral olmak üzere üç çeşit pattern vardır.

1-Creational Patterns

1.1-Abstract Factory Pattern

Somut sınıflarını belirtmeden ilişkili objelerin ailelerini oluşturmaya yarar. (Obje ailesi: birbiri ile ilişkili objeler kümesi. Örnek: Sandalye sınıfından bir obje, koltuk sınıfından bir obje ve masa sınıfından bir obje bir aile oluşturabilir. Modern tasarımda bir sandalye, bir koltuk ve bir sandalye ilişkili bir ailedir (hepsi modern tarzda) ancak örneğin, Viktoryen tarzda bir sandalye, modern tarzda koltuk ve masa ile ilişkili bir aile oluşturmaz.) Client factory sınıfından (modern veya viktoryen) haberdar olmak zorunda değildir.

1.2-Builder Pattern

Kompleks objelerin adım adım inşa edilmesini sağlar. Aynı construction kodu ile farklı tip ve temsillerde objeler yaratılmasına olanak verir. Bu inşa süreci doğrudan kontrol edilebileceği gibi, director bir sınıf aracılığı ile bazı hazır inşa tipleri de oluşturulabilir ve otomatikleştirilebilir. Bazı inşa adımları atlanabilir veya recursive olarak çalıştırılabilir. Bir builder obje tamamlanmadan bu objeyi erişilebilir kılmaz ve runtime hatalarına sebebiyet vermez.

1.3-Factory Method Pattern

Bir superclasstan obje yaratmak için ara yüz sağlar ancak subclassların, yaratılan objelerin türünü değiştirmesine izin verir. Factory method ile üretilen objelere ürün denir. Tüm ürünler superclass ile aynı ara yüzü uygulamalıdır. Client ürünlerin özelliklerini bilmez, hepsine superclassın bir objesiymiş gibi davranır.

1.4-Prototype Pattern

Sınıftan bağımsız olarak bir objenin kopyasının yaratılmasını sağlar. Klonlamaya izin veren bir obje prototip olarak isimlendirilir. Bu pattern için çeşitli şekillerde yapılandırılmış prototipler oluşturulur ve bu yapılandırmalardan birine ihtiyaç duyulduğunda ilgili prototipten bir obje klonlanır.

1.5-Singleton Pattern

Bir sınıftan sadece ama sadece bir obje yaratılmasını sağlar. Bu durum protected bir constructor ve ilk atamada objeyi oluşturan sonraki atama denemelerde aynı objeyi döndüren bir fonksiyon ile sağlanır. Bir sınıfın sadece bir objeye sahip olmasının istendiği mantıksal veya pratik durumlarda kullanılır. Örneğin meslek sınıfında türetilmiş cumhurbaşkanı alt sınıfının yalnızca bir objesi olması istenebilir. Çünkü cumhurbaşkanı aynı anda sadece bir adet bulunur.

2-Structural Patterns

2.1-Adapter Pattern

Uyumsuz ara yüze sahip objelerin iş birliği yapması için kullanılır. Örneğin XML formatındaki bir veriyi dönüştürerek json formatında işlem yapan methodlara bu veriyi vermek için kullanılabilir. Bazı durumlarda iki yönlü adapterların yazılması ve hem XMLden jsona hem jsondan XMLe dönüşüm yapılması da olasıdır. Bu mantık hem obje hem de sınıf düzeyinde kullanılabilir (Sınıf düzeyinde kullanılması için multiple inheritence destekleyen bir dil gereklidir örnek: C++).

2.2-Bridge Pattern

Büyük bir sınıf veya birbiriyle yakın ilişkili sınıflar kümesini iki ayrı hiyerarşiye (abstraction (GUI gibi düşünülebilir) ve implementation (API gibi düşünülebilir) ayırmak için kullanılır. Abstraction, implementationdan obje referansı alıp işlemlerin ara yüzde gerçekleşmesini sağlarken implementation, gerçek anlamda işlemlerin gerçekleştiği kısımdır. Bu iki hiyerarşi birbirinden bağımsız olarak geliştirilebilir. Ara yüzlerinin aynı olması gerek yoktur. Hatta genellikle implementation ilkel, abstraction daha yüksek seviye operasyonlar (implementationın ilkel operasyonlarını temel alarak) için kullanılır. Birbiriyle alakasız boyutlara sahip bir sınıfı bölmek veya runtimeda implementationlar arası geçiş yapmak için kullanılabilir. Özellikle cross-platform veya çoklu database serverlarını desteklemek için kullanışlıdır.

2.3-Composite Pattern

Objeleri tree structurelar içinde oluşturmayı ve daha sonra bu structurelar ile bağımsızmış gibi işlem yapabilmeyi sağlar. Yalnızca tree-like structurea sahip projelerde kullanılabilir. Tüm objeler aynı ara yüzü kullanmak zorundadır. Client structureların içeriğinden haberdar olmayabilir.

2.4-Decorator Pattern

Bir sınıfın (veya birden fazla sınıfın) objelerini, decorator sınıfın objesi ve fonksiyonları ile Wrap etmeye ve bu sayede objeye yeni özellik ve metotlar katmaya dayanır. Örneğin bir objeye özel bir fonksiyon işlemi yapmak istenmekte ancak sınıfın diğer objelerine bu fonksiyon uygulanmak istenmemektedir. Decorator yöntemi ile bu fonksiyon oluşturulup ardından istenen obje üstünde uygulanabilir. Sınıf üstünde değişiklik yapmadan objelere özellik yüklememizi sağlar. Ayrıca

decoratorlar decorator da Wrap edebilir. Bu sayede bir dizi işlem tek hamlede yapılabilir ve daha derli toplu kod yazımı sağlanır.

2.5-Facade Pattern

Bir kütüphane, framework veya başka herhangi bir karmaşık sınıf kümesi için basitleştirilmiş bir ara yüz sağlar. Bu kısıtlayıcı durumlara yol açabilir ancak facade pattern büyük bir kütüphanenin yalnızca bir kısmına ihtiyaç duyulması ve benzeri durumlarda kullanışlıdır. Bir projede birbirinden bağımsız birden fazla facade olabilir ve hepsi hem client hem de diğer facadelar tarafından çağrılabilir.

2.6-Flyweight Pattern

Ortak özelliklerin birden fazla obje tarafından beraber kullanılması (tüm objelerin tüm özelliklere sahip olması yerine) sayesinde RAMin daha verimli kullanılmasını ve aynı alana daha çok obje sığmasını sağlar. Objelerin intrinsic state (esas durum, değişmez) ve extrinsic state (geçici durum, zamanla değişir) olmak üzere iki tür özellikleri vardır. Yalnızca intrinsic stateleri tutan objelere flyweight denir. Extrinsic stateler çoğu durumda bir depo objesine taşınır ve gerektiğinde oradan alınır. Flyweight objelerin public üyeleri veya setter fonksiyonları olmamalıdır.

2.7-Proxy Pattern

Bir Proxy başka bir objeye erişimi kontrol eden başka bir objedir. İstek orijinal objeye ulaşmadan önce veya sonra birtakım eylemler gerçekleştirilmesine olanak sağlar. Fazla yer kaplayan ancak programın sadece belli kısımlarında ihtiyaç duyulan objelerin oluşturulmasını kontrol etmek için kullanılabilir. Ayrıca erişim kontrolü sağlayabilir, loglama için kullanılabilir.

3-Behavioral Patterns

3.1-Chain of Responsibility Pattern

Bir talebin bir handlerlar zinciri boyunca iletilmesi ile oluşur. Her handler talebi değerlendirmeye veya kendisinden sonraki handlera iletmeye (veya ikisine birden) karar verir. Her handler bir iş yapabileceği gibi, talep tek bir handler tarafından değerlendirilip diğer tüm handlerlar tarafından iletilebilir. Handlerlar bazı durumlarda talebi reddedip zinciri kırabilir. Tüm handlerlar aynı ara yüze sahip olmalıdır. Handlerların sırası ve kümesi runtimeda değişebilir.

3.2-Command Pattern

Bir talebi, talebin tüm bilgilerini içeren bir objeye dönüştürmek için kullanılır. Bu sayede talep methodlara argüman olarak verilebilir, geciktirilebilir veya bazı normalde mümkün olmayan işlemler gerçekleştirilebilir. Ayrıca yapılan işlemler tersine döndürülebilir, geri alınabilir.

3.3-Interpreter Pattern

Bir dilde tanımlı olmayan cümleleri yorumlamak için tanımlanan temsillerdir. Örneğin 9 rakamı, Romen rakamları ile IX şeklinde ifade edilir ancak bunun c++ta bir karşılığı yoktur. Interpreter ile IX, 9'a çevrilerek yorumlanabilir. Düzgün bir sınıf hiyerarşisine sokulduğunda örneğin binler basamağına kadar tüm Romen rakamları bu şekilde birkaç sınıf ile yorumlanabilir.

3.4-Iterator Pattern

Bir koleksiyonun elemanları arasında, koleksiyonun türünü açığa çıkarmadan (list, tree, stack vb) gezinmeyi sağlar. Kompleks yapıların karmaşıklığını cliente belli etmeden bu yapılar içerisindeki elemanlara erişmeyi mümkün kılar. Daha önceden bilinmesi mümkün olmayan yapı türlerine erişim sağlar. Iterationlar paralel olarak gerçekleştirilebilir, geciktirilebilir, bekletilebilir.

3.5-Mediator Pattern

Objeler arasındaki bağımlılığı azaltmak ve aralarındaki iletişimi mediator objesi üzerinden gerçekleştirmeye zorlamak için kullanılır. Bu sayede sınıfların güncellenmesi ve değiştirilmesi diğer sınıfları bozmaz, bileşenler diğer bileşenleri etkilemeden birden fazla kere kullanılabilir. Mediator'ın tanrı objesine (çok fazla method, değişken, tür vb. içeren nesne) dönüşmemesine dikkat edilmelidir.

3.6-Memento Pattern

Bir objenin eski durumunu, hakkındaki detayları açığa vurmadan kaydetmeye ve onarmaya yarayan design patterndir. Memento objeleri, eski durumunu taşıdığı obje dışındaki objeler tarafından erişilemez. Genellikle memento objelerini yönetmesi için ayrı bir sınıf oluşturulur.

3.7-Observer Pattern

Birden fazla objenin, gözlemledikleri objede meydana gelen olaylar hakkında bildirim alması için bir abonelik sistemi oluşturmaya yarar. Gözlemlenen obje publisher, gözleyen objeler ise subscriber olarak adlandırılır. Publisher bir subscriber listesine sahiptir ve bu listedeki objelere, durumunda değişiklik olduğunda bilgi gönderir. Bu sayede runtimeda objeler arası ilişkiler kurulabilir.

3.8-State Pattern

Objelerin belirli statelere geçmesini ve her state içerisinde farklı davranışlar sergilemesini sağlar. State patternla oluşturulan bir obje sanki sınıfını değiştiriyormuş gibi düşünülebilir. Her değişimle birlikte (birbiriyle çelişmeyecek şekilde (örnek: önceden olmayan bir fonksiyonu çağırmak)) objeye farklı özellikler kazandırılabilir (fonksiyonun davranışını değiştirmek gibi). State patternı uygulamak için bir kısıtlama yoktur ancak az sayıda durum için bu patterni uygulamak abartı ve gereksiz olabilir.

3.9-Strategy Pattern

Hepsi ayrı sınıflarda bulunan bir algoritma ailesi oluşturup bu sınıfların objelerini değiş tokuş edilebilir yapmaya yarayan design patterndir. Bu sayede bir obje duruma göre runtimeda algoritmanın farklı varyantları arasında geçiş yapabilir. Clientlar stratejiler arasındaki farklardan haberdar olmalıdır.

3.10-Template Method Pattern

Superclass içerisinde bir algoritma iskeleti oluşturulup daha sonra bu iskeletin alt sınıflar tarafından yapısı değiştirilmeden belirli parçalarının override edilmesine dayanan design patterndir. Algoritma adımlara bölünür ve ortak olan adımlar tüm alt sınıflarda ortak olarak kullanılır. Bu sayede kod tekrarının önüne geçilir.

3.11-Visitor Pattern

Algoritmaları işledikleri objelerden ayırmaya yarayan design patterndir. Bir sınıf hiyerarşisinde, bir davranışın bazı sınıflarda makul iken diğer sınıflarda anlamsız olduğu durumlarda kullanılır. Kompleks bir obje yapısının tüm objelerine uygulanabilir. Ancak bazı durumlarda visitorlar objelerin private üyelerine ulaşamayabilir ve sorunlara yol açabilirler.

Berke Can GÖKTAŞ

21.11.2021