**Strategy (Strateji) Kalıbı (Pattern)**

• **Intent**: Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it.

• **Amaç**: Bir algoritma ailesi tanımla, algoritmaların her birini sarmala ve onları birbirleri yerine geçebilecek hale getir. Strateji, algoritmanın onu kullanan istemcilerden bağımsız olarak değişebilmesini sağlar.

**PROBLEM**

• Yazılım geliştirirken sıklıkla bir işi yapmanın birden fazla yolunun olduğu durumlarla karşılaşırız.

• İş kuralları süreçleri yönlendirir ve süreçlerde dallanmalar olur.

• Bu farklı yollar bazen tanımlı algoritmalar olarak karşımıza çıkar bazen ise basitçe farklı yöntemler olur.

• Banka sisteminde farklı hesap türlerinin farklı ücret algoritmaları, zaman göre değişen EFT vb. ücretleri vardır,

• Sigortacılıkta farklı teminatların farklı fiyat hesapları vs. vardır,

• Lojistikte bir konteynerı (**container**) doldurmanın farklı yöntemleri olabilir,

• Farklı işlemleri yapmak için farklı yetkilendirme (**authorization**) mekanizmaları,

• Bir veriyi doğrulamanın (**validation)** farklı yolları olabilir.

• Çözüm olarak genelde yapılan, istemcinin her algoritmayı **if-elseif**

yapısının bir dalı (**ya da switch-case**) olarak ifade ederek

algoritma ailesini gerçekleştirmesidir.

• Böyle bir çözüm ise kesinlikle uzun vadeli değildir,

• Çoğunlukla tek bir algoritma olduğu düşüncesine dayanır.

• Değişmeyi göz önüne almaz.

• Öncelikle istemcilerin algoritmalardan bağımsız olması gereklidir.

• İstemcilerin algoritmayı gerçekleştirmesi, onları

• karmaşık,

• algoritmalara bağımlı ve

• bundan dolayı da kırılgan yapar.

• İstemci hem **SRP’yi** hem de **OCP’yi** bozar.

• Algoritmalar karmaşık olma ve sık değişme eğilimindedir.

• Muhtemelen sistem evrildikçe aynı işi farklı şekilde yapan algoritmaların sayısı artar, yani algoritma ailesi kalabalıklaşır.

• Birden fazla algoritmayı tek bir yerde gerçekleştirmek problemlidir,

• Pek çok **if-else if** yapısı gerekir,

• Algoritmalar arasında gereksiz gelişigüzel (accidental) bağımlılıklar oluşur,

• Sayısı artan algoritmaların bu şekilde yönetilmesi zordur, hataya açıktır.

• GoF, bir metni satırlara bölen (**linebraking**) bir yapıyı örnek olarak verir.

• Metnin tipine ve formatına göre farklı bölme algoritmaları vardır.

• İstemcinin algoritmaları gerçekleştirmesi problemlidir.

• Ayrıca yeni algoritmaların sisteme eklenmesi de istemcide değişiklikgerektirir.

• Algoritmalar istemciden bağımsız bir şekilde yönetilmeli ve değişlebilmelidir.

**SONUCLAR**

• **Strategy** değişmeyi yönetmek için sık kullanılan bir kalıptır.

• **Strategy** kalıbı ile bir algoritma ailesi istemciden bağımsız bir şekilde ifade edilir.

• İstemci **SRP** ve **OCP’ye** uygun hale gelir.

• Bu kalıp ile programlarda sıklıkla gördüğümüz **if-else-if, switchcase** gibi seçim (**conditional**) ifadeleri (**if-hell**) ortadan kalkar.

• İstemci her halukarda stratejilerin polymophic referansına sahiptir.

• Farklı algoritmalar arasındaki seçimi bağlam nesnesi yapar.

• Bundan dolayı bağlam nesnesi hangi durumda hangi algoritmayı seçeceği gibi detay bilgilere sahip olmak zorunda olabilir.

• Bazen istemcinin hangi algoritma çözümlerinin olduğunu bilmesi gereklidir.

• Böyle durumlarda istemcinin bağımlılığı artar.

• **Strategy** kalıbının negatif sonucu olarak şunlar sayılabilir:

• Her algoritma farklı bir sınıf ile ifade edildiğinden sınıf ve nesne sayısı artar.

• İstemci ile bağlam (context) nesnesi arasındaki iletişim artar.

• Kurumsal uygulamalarda farklı durumlarda farklı şekillerde işletilen süreçler, hesaplama ve algoritma şeklinde ifade edilen iş kuralları için **Strategy** kalıbı çok uygundur.

• Örneğin bir hesaplamanın müşteri tipine bağlı olarak yapıldığı durumlarda, hesaplamaları tek bir metotta **if-else-if ya da switch-case** içine koymayıp, **Strategy** kalıbıyla her birini ayrı sınıflarda ifade etmek, çok daha temiz ve yönetilebilen kod sağlar.

**• Dolayısıyla ne zaman bir şeyi hesaplamanın farklı yollarıyla karşılaşırsanız, bir iş kuralının bir faktöre göre değişen şekilleri vs. varsa Strategy kalıbını kurun.**

• **O anda sadece tek bir yol olsa bile bunu yapın çünkü muhtemelen iş evrildikçe birden fazla yolu ortaya çıkacaktır.**

• **Bu noktada uzun vadeli ve stratejik (hikmetli (wise), ferasetli, uzak görüşlü (prudential)) düşünün.**

**REFACTORING**

• **Strategy** kalıbı sıklıkla refactoringe konu olur.

• Çünkü metotların sahip olduğu karar mekanizmaları, yazılım geliştirme hatta bakım sürecinde karmaşıklaşır.

• Aslen alternatifsiz olan bir hesap ya da algoritma, zamanla alternatifler kazanamaya başlar ve bu durum, ilk başta düşünülmeyen **Strategy** kalıbını gündeme getirir.

• **M. Fowler, Refactoring** aşağıdaki konularıda iyileştirici olarak

**Strategy** kalıbını sayar:

• Switch statements

• Replace Type Code with State/Strategy

• Replace Conditional with Polymorphism

**KULLANIMLAR**

• Programlama dillerinin, bileşen ve çerçevelerin APIlarında **Strategy**

sıklıkla kullanılır.

• Örneğin GUI yapılarında kapların (container) yerleştirme algoritmaları

birer **Strategy** nesnesidir.

• **Java’da java.awt paketindeki FlowLayout, BorderLayout,**

**GridLayout ve diğer pek çok yerleştirme nesnesi LayoutManager**

**arayüzünü gerçekleştirir.**

**Diğer Kalıplarla İliskisi**

• GoF kalıpları arasında bir arayüzü gerçekleştiren alt sınıflar sıklıkla kullanılır.

• Bu açıdan yapı olarak **Strategy, Command, Proxy, Abstract Factory**, **State** gibi kalıpların birbirlerine benzedikleri düşünülür.

• Gözden kaçırılmaması gereken şey, kalıpların amaçları ve bileşenlerinin rolleridir.

• Örneğin **Abstract Factory**, nesne oluşturma açısından **Strategy** gibi görülebilir.

• **Abstract** **Factory** bir nesne ailesini oluştururken **Strategy** algoritma ailesini yönetir.

• Yapısal olarak benzese de **Strategy** ve **Proxy** farklı çözümlerdir, **Proxy’de** saklanan nesne ile onun vekili bir algortimanın farklı gerçekleştirmeleri değildir.

• **Strategy**, **Command’in** özel bir hali olarak görülebilir.

• **Strategy** nesneleri **Flyweight** gibi görülebilir.

**UYGULAMA**

**•** Adapter kalıbındaki 3 ayrı sıralayıcıyı Strategy kalıbıyla yapılandırın:

• **BubbleSorter, JavaArraysSorter & ApacheSorter**

