**FAÇADE KALIBI**

• **Intent**: Provide a unified interface to a set of interfaces in a subsystem. Façade defines a higher level interface that makes the subsystem easier to use.

• **Amaç**: Bir alt sistemdeki arayüzlere, bir birleşik arayüz sağla. Façade, alt sistemi daha kolay kullanmak için daha yüksek seviyeli bir arayüz sağlar.

• Bu kalıbın ilk ismi “**Glue**”ydı sonra **Façade** olarak değiştirildi.

• Fransızca aslında “Façade” binaların ön yüzü, ön cephesi anlamına gelir.

• Zaten kelime yapısı itibarıyla "face" yani "yüz" ile de akrabadır

**PROBLEM**

• Sistemler tabiatıyla karmaşıktır ama arayüzlerinin basit olması beklenir.

• Çünkü istemcilerin kullandıklarına olan bağımlılıkları en aza indirilmelidir.

• Sistemlerin karmaşıklığını yönetmek için de parçalayarak onları alt sistemlere bölmek anlamlıdır.

• Bu durumda alt sistemler birbirlerinin istemcisi olur.

• Dolayısıyla alt sistemler arasındaki bağımlılıkları da olabildiğince asgari

seviyede tutmak gerekir.

• GoF örnek olarak bir derleyici (compiler) alt sistemini ele alır.

• Bu alt sistemde kodu okuyan, pars eden, söz dizimi kontrolleri yapan, son kodu üreten vb. pek çok sınıf vardır.

• İstemciler zaman zaman bu özel iş yapan yapılara ulaşmak isteselerde genelde pek çok istemci, derleyici alt sistemini basit bir şekilde kullanmak ister.

• Bu türden istemciler için alt sistemdeki nesnelerin sağladığı hizmetlerin

tek ve birleştirici bir arayüz (unified intereface) üzerinden açılması

gerekir.

• Tek erişim noktası (single point of contact)

• Bu istemcilerin de onları geliştirenlerin de işini kolaylaştırır.

• Bu ihtiyaca uygulamaların yapısı üzerinden de yaklaşmak mümkündür:

• İş mantığı çoğunlukla süreç (process) yapısındadır ve süreç pek çok farklı

nesneden hizmet alarak ilerler.

• Süreçler karmaşıktır ve hangi nesneden hangi durumlarda hizmet

alınacağı bilgisini detayıyla ifade eder.

• Süreç aynı zamanda pek çok iş kuralı da içerir.

• Böyle durumlarda istemci nesnelerin süreci tüm detayıyla bilmeleri

istenmez.

• İş mantığının detayını istemcilerin bilmesi onları karmaşık ve kırılgan

yapar.

• Bu, farklı istemcilerin iş detayını tekrarlamalarına da sebep olur.

• Bu da istemciler için bakım problemi demektir.

• Bu, istemcilerin teknolojik detayları bilmesini de gerektirebilir.

• Ayrıca iş süreçlerinin detayı, tamamen dış istemciler olması halinde,

dışarıya açılamayacak kadar önemli ve gizli olabilir.

• Dolayısıyla detaylara değil arayüzlere bağımlı olma prensibi, alt

sistemlerin birbirleriyle olan ilişkilerinde de geçerli olmalıdır.

bu görsel solution paketi icinde notlar icindir.

**SONUCLAR**

• Karmaşık bir alt sistemi basit bir şekilde kullanmak için **Façade** güzel bir çözümdür.

• **Façade**, alt sistemlere bir giriş noktası sağlar.

• **Façade**, alt sistemlerin arayüzü (interface) olur.

• **Façade**, pek çok ince (fine-grained) arayüzden oluşan bir sistemi, kalın (coarse-grained) bir arayüze dönüştürür.

• Nasıl sınıflar birbirlerine arayüzleri üzerinden bağımlı olurlarsa ve bu arayüzleri olabildiğince minimal yapmak gerekliyse bir sistemin alt parçaları da birbirlerine minimal arayüzler üzerinden bağımlı olmalıdır.

• Bu şekilde alt sistemlerin birbirlerinin iç yapılarına bağımlı olmaması sağlanır.

• Ayrıca istemci ile alt sistemin parçaları arasındaki bağımlılık da ortadan kalkar.

• Gerektiğinde istemcinin alt sistemin parçalarına ulaşması söz konusu olabilir.

• Façade nesnesinin kullanımı zaman zaman seçilen mimari ve teknolojilerden dolayı da faydalı hatta zorunlu olabilir.

• Örneğin uzak nesnelere (distributed objects) erişerek hizmet alma durumunda, bu nesnelere pek çok ufak-tefek iş yapan çağrı yapmaktansa, uzak nesnelerin önüne konulacak ve kalın (coarsegrained) metotlara sahip bir uzak Remote Façade nesnesi ile, yapılacak uzak metot çağrılarının sayısı bir hayli azaltılabilir.

• Bu özelliklerinden dolayı **Façade’ın** birlikteliği (cohesion) düşük, bağımlılığı (coupling) yüksek olma eğilimindedir.

• Çünkü **Façade** gizlediği alt sistemin pek çok nesnesi ile haberleşir.

• Bu amaçla birden çok façade nesnesi kullanılabilir, hatta façadelar da katmanlı olabilir.

• Bu şekilde **Façade’ın** birlikteliğinin çok düşmemesi, bağımlılığının da çok artmaması hedeflenir.

• **Façade’ın** metotları, süreci başlatan, yönlendiren ve sonlandıran mahiyette olurlar.

• **Façade** nesneleri de genelde durumsuz (stateless) nesnelerdir aksi taktirde durumlu (stateful) **Façade** nesnelerinin sayısı istemci sayısına paralel olarak artar ve bu da ölçeklenirlik (scalability) problemleri ortaya çıkarabilir.

• **Façade**, kurumsal uygulamalarda, sunucu tarafından hizmet alan, web, web servis, uzak vb. arayüzün, istemcinin sunucuya ulaştığı ilk nokta olması açısından da önemli bir role sahiptir.

• Dolayısıyla **Façade**, sunucu tarafının arayüzü olarak hizmet eder.

* Muhtemelen bu durumda birden fazla **Façade** nesnesi gereklidir.

• **Façade**, kurumsal uygulamalardaki katmanlar (tiers/layers) arasındaki ilişkiyi basitleştiren en temel kalıptır.

• Dolayısıyla ne zaman karmaşık bir sistemin kullanımını basitleştirmek, ona basit bir arayüz sağlamak isterseniz **Façade** kullanın.

• Ne zaman karmaşık bir sisteme, az metot çağrısı ile çok iş yapacak şekilde erişmek isterseniz bu sisteme bir **Façade** tasarlayın.

• Katmanlı yapı kullanırken, her katman için bir giriş noktası olarak **Façade** kullanın.

**DİĞER KALIPLARLA İLİŞKİ**

• **Abstract Factory**, nesne yaratması açısından bir nesne yaratma alt

sistemi **Façade’dı** olarak görülebilir.

• **Façade**, **Singleton** olabilir.

• Ama bu durumda **Façade** nesnesinin durumunu thread-safe yapmak

gerekir.

• Bu amaçla **Façade** nesnesi durumsuz (stateless) hale getirilebilir.

• **Façade** ile **Adapter** arasındaki fark, ilkinin karmaşık arayüzleri,

basit olanla değiştirmesine karşın Adapter’in, uyumsuz arayüzü

uyumlu hale getirmesidir.

• **Façade** karmaşıklık ve bağımlılığı çözmeye çalışır.

**UYGULAMA**

**• Bir sigortacılık sisteminde prim hesabı, pek çok nesne üzerinde**

**çağrılan metotlardan oluşan karmaşık bir süreçtir.**

**• Prim hesaplama süreci kabaca şu adımlardan oluşur:**

**• Risk bilgilerini getir,**

**• Teminatları getir,**

**• Poliçe teminatlarını belirle,**

**• Teminat fiyatlarını hesapla,**

**• İndirimleri hesapla,**

**• Surprimleri hesapla,**

**• Primi hesapla,**

**• Bu yapıyı Façade kalıbıyla modelleyin ve kodlayın.**

**• Bu amaçla uygun nesneleri oluşturun.**