**Chain Of Responsibility**

• **Intent**: Avoid coupling the sender of a request to its receiver by giving more than one object a chance to handle the request. Chain the receiving objects and pass the request along the chain until an object handles it.

• **Amaç**: İsteği yerine getirmek üzere birden fazla nesneye şans vererek, isteği gönderenin, alana olan bağımlılığından kaçın. İsteği alan nesneleri zincire koyup, isteği, yerine getiren nesneyi buluncaya kadar zincir boyunca dolaştır.

**PROBLEMS**

• Sıklıkla isteği gönderen nesneler ile isteği yerine getirecek nesneler arasındaki bağımlılığı azaltmak isteriz.

• Bu amaçla, istemci ile isteği yerine getiren nesneler arasına başka soyutlamalar konabilir.

• Böyle bir ihtiyacı daha önce **Command** kalıbıyla karşılamıştık.

• **Command** kalıbında isteği yerine getiren farklı emir nesneleri arasında seçimi yapacak merkezi bir nesneye ihtiyaç duyulabilir.

• Ama bazen isteği kimin yerine getireceğinin belirlenmesini merkezileştirmekten kaçınmak gerekebilir.

• Bu durumda isteği kimin yerine getireceği, doğrudan isteği yerine getirecek hedef nesneler tarafından belirlenir.

• Hatta aynı istek, birden fazla nesne tarafından ortaklaşa bir şekilde de yerine getirilebilir.

• Bu yaklaşımda hedef nesneyi bulmayı merkezi bir sorumluluk olmaktan çıkarmak ve sorumluluğu hedef nesnelerin kendisine yaymak, biraz pahalı gelse de, etkin bir yöntemdir.

• GoF, bağlama duyarlı yardım (context-sensitive help) örneğini verir.

• Burada kullanıcı kullandığı GUI üzerinde herhangi bir yerde iken yardım almak isteyebilir.

• Yardımı sağlayacak olan nesne, GUI üzerinde seçilen elemana ve bağlama bağımlıdır.

• Buna göre birden fazla yardım nesnesi birden fazla yardım konusu sağlayabilir.

• Yardım konuları, dolayısıyla yardım nesneleri, daha özel (specific) olandan daha genel (generic) olana doğru hizmet vermelidir ve sonuçta bir ya da daha çok yardım nesnesinin yardım bilgisi görüntülenmelidir.

**SOLUTION**

• İstemci ile isteği yerine getiren nesneler arasındaki bağımlılığı azaltmanın yollarından birisi **Chain of Responsibility** kalıbıdır.

• Bu amaçla bu kalıp bütün bir nesne hiyerarşisini bir arayüz arkasına saklamakla kalmaz, kendisini yerine getirecek nesneyi bulması için isteği hiyerarşideki nesneler arasında gezdirir.

• **Chain of Responsibility** kalıbında, işi isteyen nesneler de işi yerine getirecek yani hedef nesneler de bir hiyerarşide bulunurlar.

• Ayrıca hedef nesnelerin, işi isteyen nesnelerle doğrudan iletişimde bulunmasını engellemek amacıyla hedef nesneler bir zincir boyunca, en basit olanından en karmaşık olanına doğru sıralanırlar.





• Hedef nesneler arasında bir zincir oluşturmak için zincirdeki her nesnenin bir sonraki nesneyi bilmesi sağlanır.

• Bu amaçla kendi tipinden nesne olan kurucu ya da set metodu gereklidir.

• **İstenen iş, önce zincirdeki ilk nesneye verilir,**

**• Eğer nesne verilen işin kendi sorumluluğuna uygun olduğuna karar verirse işi yerine getirir,**

**• Aksi taktirde işi zincirdeki bir sonraki nesneye geçirir.**

**• Bu şekilde iş, yerine getirecek nesne bulununcaya kadar, zincir boyunca iletilir.**

**• Bazen bir istek birden fazla nesne tarafından da işlenebilir.**

• Zincirdeki nesnelerin kendilerine gelen isteği karşılayıp

karşılayamayacaklarına karar vermeleri gerekir.

• Bu amaçla gelen istekte ya da istekte bulunan nesnede ayrıt edici bir

durum olmalıdır.

**SONUCLAR**

**• Şu durumlarda Chain of Responsibility kalıbı kullanılabilir:**

**• Eğer bir isteği birden fazla nesne yerine getirebiliyor ve hangisinin yerine getireceği önceden bilinmiyorsa,**

**• Ya da bilinse bile isteyen ile isteği yerine getiren arasında statik bir ilişki kurmak istenmiyorsa yani isteği kimin yerine getireceği dinamik olarak belirlenecekse,**

**• Ya da isteği kimin yerine getireceğinin merkezi bir yapıda belirlenmesi istenmiyorsa,**

**• Bu kalıp ile istemcinin, istekleri yerine getiren nesnelerle aralarındaki**

**bağımlılık azalır.**

**• Bu şekilde işi yerine getiren nesneler arasında farklı yapılar kurulabilir**

**ve istemci bunlardan habersiz olur.**

• Bu yaklaşımın negatif tarafları şunlardır:

• Zinciri kurmak karmaşık olabilir,

• Sorumluluk zinciri düzgün kurulmadığında, yerine getirilmeyen istekler olabilir.

• İsteklerin yerine getirilmesinden sonra, istemciye çıktı döndürmek ya da bir siteği birden fazla nesnenin yerine getirmesi gibi durumları yönetmek kolay değildir.

**KULLANIMLAR**

• Programlama dillerinde sıra dışı (exception) durum mekanizmaları Chain of Responsibility kullanır:

• Çalışma zamanında fırlatılan Exception nesnesi, bu nesneyi fırlatan metottan başlayarak çağrı zincirinde (**call** **chain**) geriye doğru giderek uygun handler nesnesini arar.

• Fırlatılan sıra dışı durum nesnesinin tipine uyan bir handler bulunduğunda arama durur ve sıra dışı durumun gereği orada yerine getirilir.

• Web uygulamalarındaki filtreleme yapısı da **Chain of Responsibility** kullanır.

• Java’da Servlet API’sinde ve ASP.NET’de filtre mekanizması bu kalıp ile kurulmuştur.

**UYGULAMA**

**Bir yazılımın bakımından sorumlu ekibe gelen istekler, bug, UI change,**

**improvement ve project olarak kategorize edilmektedir.**

**• Gelen istekler, bir istek bandına konulmakta ve isteğe uygun**

**developerlar tarafından banttan alınıp işlenmektedir.**

**• Bu yapıyı Chain of Responsibility kalıbı ile gerçekleştirin.**