**Visitor ( Ziyaretçi ) Pattern (Kalıbı)**

• **Intent**: Represent an operation to be performed on the elements of an object structure. Visitor lets you define a new operation without changing the classes of the elements on which it operates.

• **Amaç**: Bir nesne yapısının parçaları üzerinde çalışacak bir operasyon ortaya koy. Visitor, parçaların sınıflarında değişiklik yapmadan onlar üzerinde çalışacak yeni bir operasyon tanımlamanıza izin verir.

• Bu kalıbın ilk ismi “**Walker**”dı sonra **Visitor** olarak değiştirildi.

**PROBLEMS**

• Zaman zaman nesnelerin yeni davranışlara sahip olması gerekir.

• Bazen gerekli olan yeni davranışlar birbirleriyle ilgilidir dolayısıyla bir grup oluşturur, yani nesnelerin bu gruptaki davranışlara sahip olması beklenir.

• Bazen gerekli olan bu davranışlar nesneden nesneye de değişebilir.

• Bir sınıfın alt tiplerinin farklı yeni davranışlaa sahip olması gibi.

• Sıklıkla yeni davranışa sahip olan nesneler bir hieyrarşi içinde olurlar ve **Composite** gibi kalıplardaki gibi pek çoğu bir başka nesnenin parçasıdır.

• Böyle durumlarda bu davranışları nesneler üzerinde tanımlamak iki sebepten dolayı sağlıklı olmaz:

• Böyle bir yaklaşımda, parçalar çok farklı davranışlarla kirlenir, birliktelikleri (cohesion) düşer ve parçalar gittikçe büyür,

• Her yeni davranış ihtiyacı doğduğunda parçaların değişmesi gerekir.

• Bu da ya var olan parça sınıflarında sıklıkla değişiklik yapmak ya da her yeni davranış için yeni bir alt sınıf yaratmak anlamına gelir.

• **GoF örnek olarak, programları soyut sözdizimi ağacında (abstract**

**syntax tree) sunan derleyiciyi (compiler) ele alır.**

• Derleyici bu ağaçtaki öğeler için semantic analiz yapar ve son kodu üretir.

• Derleyici bu amaçla tip kontrolü, akış analizi, değişkenlerin ilk değer kontrolü, kod optimizasyonu vb. işlemler gerçekleştirir.

• Bu işlemlerin herbiri, kodun atama ifadeleri, değişkene erişim vb. kısımalrında farklı şekillerde çalışır.

• Dolayısıyla farklı kod parçalarını temsil eden **Node** sınıfları ve sınıflarda bu işleri yapan metotlar vardır.

• Bu çözümdeki en temel problem, kodun farklı parçalarını emsil eden **Node** nesnelerinin aynı zamanda **Node’lar** üzerinde çalışacak metotlara da sahip olmasıdır.



• Bu şekilde aynı işin farklı Node sınıflarına dağıtılması, anlaşılması ve değiştirilmesi zor bir sistem ortaya çıkarır.

• Çünkü farklı iş yapan metotlar aynı sınıfta bulunur.

• Üst tipe yapılan bir değişiklik tüm alt sınıfların tekrar derlenmesini gerektirir vs.

**• Her işin ayrı bir sınıfta ele alınması ve Node sınıflarının bu işlerden bağımsız olarak modellenmesi daha rahat bir sistem ortaya koyabilir.**

**SOLUTION**

• Böyle durumlarda yeni davranışları nesnelerin üzerinde tanımlamak yerine, bir başka ziyaretçi nesnede tanımlamak ve bu ziyaretçinin davranışlara ihtiyaç duyan nesneleri ziyaret ederek davranışları gerçekleştirmesine izin vermek daha uygun bir çözüm olur.

• Davranışlara özel olarak geliştirilmiş ve nesneleri ziyaret ederek davranışları onlar üzerinde gerçekleştiren nesneye **Visitor** denir.

• Bu çözümde ihtiyaç duyulan yeni davranışlar, sadece o iş için oluşturulmuş **Visitor’da** tanımlanır.

• Ziyaretçi, nesneler üzerindeki **accept**() metoduna geçilir.

• Bundan dolayı nesne hiyerarşisinin en üstünde ziyaretçi nesneyi kabule den **accept**() metodu tanımlanır.

• **accept**() metoduna sahip nesne de kendisini ziyaretçinin **visit**() metoduna geçer.

• Ziyaretçi üstünde her farklı alt tip için ayrı bir **visit**() metodu vardır.

• Bu şekilde nesneleri ziyaret eden **Visitor**, gerekirse nesnelerin durum bilgilerini kullanarak davranışları yerine getirir.

Nihayetinde Visitor, aksi taktirde nesnelerin üzerinde tanımlanacak

davranışlara sahip olan ve bu şekilde nesnelerin kirlenmesini önleyen

apayrı bir nesnedir.

**SONUCLAR**

• **Visitor** kalıbı ile var olan nesnelere yeni davranışlar eklemek çok daha

kolay hale gelir.

• **Visitor** kalıbı sayesinde birbirleriyle ilgili davranışlar bir arada toplanır.

• Böylece iş nesneleri, ilgisiz pek çok davranışa doğrudan sahip

olmazlar, bu davranışları dışarıdan alırlar (**outsourcing**).

• **Visitor** ziyaret ettiği nesnelerin durum bilgilerine ihtiyaç duyabilir.

• Bu amaç için parçalarda get metotlarına ihtiyaç vardır.

• **Visitor** ziyaret ettiği nesnelerin, set metotlarıyla durum bilgilerini değiştirebilir.

• Bu durum **Visitor’ın** sarmalamayı (**encapsulation**) bozduğu şeklinde ifade edilir.

• **Visitor** kullanımı, var olan nesnelerde değişiklik gerektirebilir.

• **Visitor** nesnesinin işini yapabilmesi için, ziyaret edeceği nesnelerin

uygun durum ve davranışlara sahip olup olmadığı, sahip değilse bu

durumun nasıl düzeltileceği kritik bir konudur.

• Aslolan nesneyi olabildiğince değiştirmeden yeni **Visitor** nesneleri kurgulamaktır.

• **Visitor**, aralarında bir hiyerarşi olmayan, ilgisiz nesneler üzerinde

çalışamaz.

• Çünkü **Visitor** nesnesi, ziyaret edeceği nesnelerin tipini derleme

zamanında bilmelidir.

• Dolayısıyla **Visitor** nesnesi hiyerarşiye özeldir.

• Eğer bir torbada bulunan nesneler, ziyaret edecek ise Visitor’ın

**iteration** işlemine ihtiyaç vardır.

• Iteration bir kaç farklı nesnenin sorumluluğu olabilir.

• Visitor’ın kendisi iteration yapabilir.

• Parçalara sahip olan bileşik nesne yapabilir ki factory örneğindeki

durum da budur.

• Ya da iterasyon için özel bir nesne oluşturulabilir.

**DİĞER KALIPLARLA İLİSKİ**

**Composite kalıbı ile, Visitor’ın uygulanması için gerekli bileşik**

**nesneyi yaratılabilir.**

**• Iterator kalıbı Visitor’ın parçalara ulaşması için kullanılır.**

**UYGULAMA**

**Visitor kalıbını kullanarak, bir bankanın sahip olduğu Account**

**hiyerarşisindeki tüm nesnelere, geçilen bir ortama print edilebilme**

**yeteneğini nasıl kazandırırsınız?**