**Bridge(Köprü) Kalıbı (Pattern)**

• **Intent**: Decouple an abstraction from its implementation so that the two can vary independently.

• **Amaç**: Bir soyutlamayı gerçekleştirmesinden ayır öyle ki ikisi birbirinden bağımsız olarak değişebilsin.

• Bir cümledeki tüm kelimeleri anlayıp da cümlenin kendisini anlamamak

işte böyle bir şeydir!

• Bir soyutlamanın gerçekleştirmesinden ayrılması ne demektir?

**PROBLEM**

• Anlaşılması en zor olan kalıplardandır.

• Çünkü bildiğimiz ve alışkın olduğumuzdan farklı bir yapı kurar.

• Ama güçlü bir kalıptır.

• Temelde şu iki prensibi vurgular:

**• Değişen ile değişmeyen kısımları ayırmak.**

**• Bileşimi, kalıtıma tercih etmek**

• Normalde soyutlamalar ile gerçekleştirmeleri birbirlerinden ayrı değildir

aralarında kalıtım yani **is-a** ilişkisi vardır.

• Soyutlama arayüzü, farklı alt tipleri ise arayüzün farklı

gerçekleştirmelerini belirler



• Fakat bu durum, soyutlama ile alt tipleri arasında ciddi bir bağımlılığa sebep olur.

• Soyutlama ile gerçekleştirmesi arasındaki bağımlılık arayüz seviyesinde kalmaz, gerçekleştirmeye de bağımlılık söz konusudur.

• Kalıtımda gerçekleştirme, soyutlamadan hem arayüz hem de gerçekleştirme devralabilir.

• Kalıtım arayüz birliği sağlar, gerçekleştirmeler asgari halde soyutlamalarla aynı arayüze sahiptirler,

• Ama soyutlamanı tipine, sınıf ya da arayüz olmasına ve gerçekleştirmenin neyi devraldığına ve ne kadar metot ezdiğine (override) göre, farklı miktarlarda gerçekleştirme devralması söz konusudur.

• Soyutlamalar ile gerçekleştirmeleri arasındaki ilişki **is-a** olarak ifade edildiğinde iki tarafı da kısıtlayan bir durum oluşur:

1. Soyutlama değişirse, gerçekleştirmeler de değişmek zorunda kalır.

• Kalıtım, en temel nesne mekanizmalarından olmasına rağmen yüksek bağımlılık oluşturduğu için bazen kaçınılması gereken bir yapı olur:

**Inheritance violates encapsulation!**

• Bu yüzden GoF’un ikinci prensibi **“favor object composition over**

**class inheritance”**dır.

2. Bazen de gerçekleştirmelerin soyutlamalara uyma zorunluluğu, onların rahatça geliştirilmelerini önler. Örneğin

• Gerçekleştirmelerin arayüzünün serbestçe değişmesini sağlamak,

• Gerçekleştirmelerin daha önceden olduğu ve arayüzlerin sonradan belirlendiği durumlar,

• Birden çok ve muhtemelen farklı zamanlarda farklı arayüzlerde yapılmış gerçekleştirmelerin aynı soyutlamanın altına konması gerektiği durumlar, vs.

**Is-A ile Has-A -I**

• Dolayısıyla soyutlamalar ile onların gerçekleştirmeleri arasındaki ilişki her zaman **is-a** olarak gösterilmek zorunda değildir.

• Bu ilişki bazen **has-a** ile de gösterilebilir.

• Ve bu duruma yüksek bağımlılığın getirdiği negatif etkiler aşılabilir.

• **is-a** olan bir ilişki **has-a’ye** döndürüldüğünde gerçekleştirmelerin

soyutlamalara olan bağımlılığı azalır, çünkü devralmak değil kullanmak

söz konusudur ve bundan dolayı soyutlamalardaki değişiklikler,

gerçekleştirmeleri etkilemez.

• Soyutlamalar rahatça değiştirilebilir:

• Arayüzleri değiştirilmez çünkü istemciler kırılır,

• Ama gerçekleştirme detayları rahatça değiştirilebilir, soyutlamaları gerçekleştiren sınıflar bundan etkilenmezler.

• Gerçekleştirmeler rahatlıkla değiştirilebilir çünkü, hem arayüz hem de

gerçekleştirme detayı olarak kendisine uymak zorunda olduğu

soyutlamalar yoktur.

• Ya da vardır ama aralarındaki ilişideki uyumluluk problemli değildir

çünkü ilişki is-a değil has-a’dir.

**SONUCLAR**

• Bir soyutlama ile gerçekleştirmesi arasındaki bağımlılığı ne zaman hafif tutmak isterseniz, **Bridge** kalıbını kullanabilirsiniz.

• Neden böyle bir şey istenir?

• Soyutlamayı değiştirmediğiniz halde gerçekleştirmelerini rahatça değiştirmek isterseniz,

• Gerçekleştirmelerini etkilemeden soyutlamaları değiştirmek isterseniz,

• Kısaca, soyutlamalar ile gerçekleştirmelerini birbirlerinden olabildiğince bağımsız kılıp, rahatça değiştirmek isterseniz

• Soyutlamayı değiştirmek istemeyiz çünkü istemciler ona bağımlıdırlar.

• Ama soyutlamaların gerçekleştirmeleri, soyutlamalardan bağımsız olarak değişmesi gerektiğinde **is-a** yapısı sıkıntı çıkaracaktır.

• Gerçekleştirmelerin arayüzlerinin ve detaylarının sıklıkla değiştiği birMortamda, sistemi kırılmalardan korumanın yolu **Bridge** kalıbıdır

• Dolayısıyla **Bridge** kalıbında soyutlamalar ile onların gerçekleştirmeleri arasında **is-a** ilişkisi yoktur.

• Ve her ikisinin de ayrı hiyerarşisi vardır.

• İki hiyerarşi arasındaki **has-a** ilişkisi, bir **Bridge** (**köprü**) ile kurulur.

• İstemci, gerçekleştirmeden habersiz bir şekilde, sadece soyutlama ile iletişimde bulunur.

• Dolayısıyla gerçekleştirmedeki değişiklikler, istemciyi etkilemez, sadece soyutlamaları etkiler.

• Bu etki de çok daha az bir bağımlılık üzerinden olur çünkü arada **is-a değil has-a** ilişkisi vardır.

• **Bridge** kalıbı, karmaşıklığı arttırma pahasına esneklik sağlar.

• Bu yaklaşımın şu ana kadar alışkın olunandan farklı olması, **Bridge** kalıbının anlaşılmasındaki temel zorluktur.

**KULLANIMLAR**.

• Eşli (peerful) çalışan yapılarda sıklıkla **Bridge** kalıbı kullanılır.

• Çünkü eşler ile onların arayüzleri arasındaki bağımlılığın getirdiği kısıtlar aşılmaya çalışılır.

• GUI sistemlerinde, var olan soyutlamalar, farklı platformlar için gerçekleştirilir.

• Veri tabanı iletişim yapılarında

• **JDBC**:

• http://www.informit.com/articles/article.aspx?p=29302

• http://www.journaldev.com/1491/bridge-design-pattern-java

• http://www.oodesign.com/bridge-pattern-object-persistence-apiexample-

java-sourcecode.html

**UYGULAMA**.

• JPA/EntityFramework gibi bir çerçeve standardı için bir torba çatısı (collection/container framework) oluşturmak istediğimizi düşünün.

• Bu çatıda torba (collection), küme (set) ve dizi (list) soyutlamalar ile soyutlamaların gerçekleştirmeleri olacaktır.

• Ama soyutlamalar, Java/C#’taki torba arayüzleriyle uyumlu olmalıdır.

• Gerçekleştirmeleri, soyutlamalarından bağımsız değişebilecek şekilde **Bridge** kalıbını kullanarak nasıl kurgulardınız?