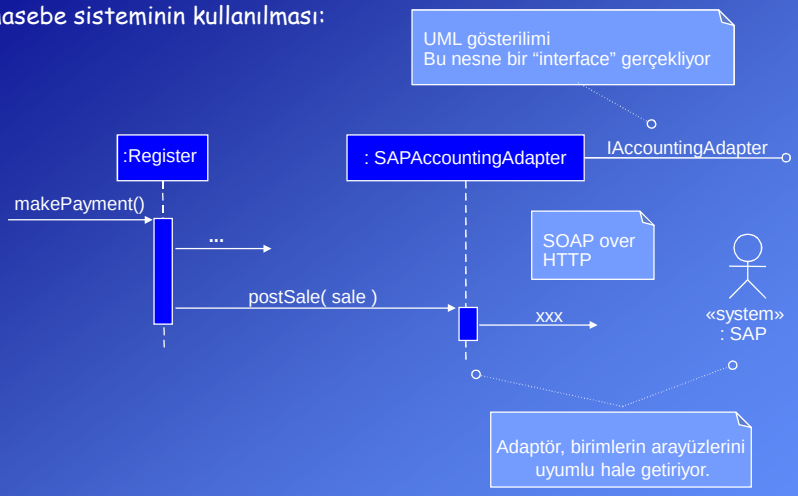
Muhasebe sisteminin kullanılması:



Karmaşık ve özel kalıplar daha temel GRASP kalıpları ile ifade edilebilirler.

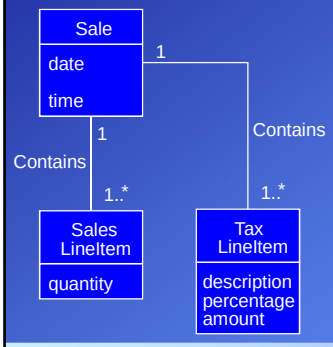
Örneğin yukarıdaki çözümde değişimlerden etkilenmeyi azaltacak şekilde dolaylılık ve çok şekillilik kullanılmıştır.

Tasarım Aşamasında Çözümleme Aşamasına İlişkin Yeni Kavramların Keşfedilmesi

Yazılım sisteminin tasarımı yapılırken, çözümleme aşamasındaki modelde yer almamış olan yeni kavramlar ortaya çıkabilir.

Örneğin adaptörlerin tasarımında oluşturulan getTaxes metodu, vergi kalemleri listesi (TaxLineItems) geri döndürür. Bu kavram (vergi kalemleri) çözümleme aşamasında modelde yer almamış olabilir.

Tasarım aşamasında keşfedilen kavramlar, eğer gerek görülürse çözümleme modeline eklenebilir. Bir kaç iterasyondan sonra çözümleme (analiz) modeli işlevini tamamlayacağı için bu ekleme gereksiz de olabilir.



Hatırlatma: Analiz modeli,
- problem domenindeki kavramları daha iyi anlayabilmek,
- yazılım sınıflarını oluştururken esinlenmek için kullanılır.

Eğer çözümleme modeli sonraki aşamalarda tekrar bir kaynak olarak kullanılacaksa yeni keşfedilen kavramların modele eklenmesi uygun olabilir. Daha çok tercih edilen yöntem ise, tasarım modelinden geri gidilerek (*reverse-engineering*) sonraki aşamalarda kullanılabilecek çözümleme modelinin oluşturulmasıdır.

2.Fabrika (*Factory*), Somut Fabrika (*Concrete Factory*)

Adaptörlerin kullanılması, bu nesneleri kimin yaratacağı problemine neden olur. Diğer bir problem ise belli bir durumda olası adaptörlerden hangisinin yaratılıp kullanılacağıdır.

Adaptörleri yaratma işi, bunu kullanacak olan yazılım sınıflarından birine atanabilir. Ancak uygulama sınıflarına bu tür sorumluluklar atamak uyumu bozar ve uygulama sınıflarının değişken adaptörlere bağımlı olmasına neden olur.

Önemli tasarım prensiplerinden biri; bir yazılım sistemindeki değişik işler yapan kısımların (katmanların) birbirlerinden bağımsız olarak ele alınmasıdır (***seperation of concerns***). Buna göre de adaptörlerin uygulama nesnelerinin dışındaki bir nesne tarafından yaratılması uygun olacaktır.

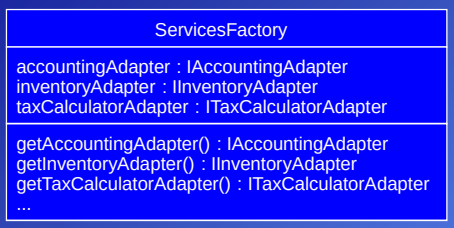
Bu bölümde GoF kalıplarından soyut fabrikanın (***Absract Factory***) daha basit bir şekli olan fabrika (*Concrete Factory*) kalıbı ele alınmıştır.

Problem: Nesnelerin yaratılmasında karmaşık kararlar vermek gerekiyorsa ve uyumluluğu arttırmak için yaratma işlemlerinin diğer işlemlerden ayrılması isteniyorsa nesne yaratılma sorumluluğu nasıl atanmalıdır?

Çözüm: Nesnelerin yaratılma sorumluluklarını yapay bir sınıf olan fabrika sınıfına atayın.

Fabrikalar sadece adaptörler için hazırlanmazlar. İlerideki konularda başka nesnelerin yaratılması sorumluluğunun da fabrika nesnelerine verildiği görülecektir.

Örneğin POS sisteminde adaptör nesnelerini yaratmak için ServicesFactory adında bir yapay sınıf oluşturulabilir.



Vergi hesabı programını kullanmak isteyen nesne gerektiğinde ServicesFactory nesnesinin getTaxCalculatorAdapter metodu çağırılacaktır.

Bu metot o andaki kriterlere göre hangi tipte bir adaptörün kullanılacağına karar verecek, eğer yoksa uygun bir adaptör nesnesi yaratacak ve bu nesnenin adresini geri döndürecektir.

Böylece vergi hesabı yapılmak istendiğinde bu adaptör ile ilişki kurulacaktır (getTaxes metodu çağrılacaktır).

ServicesFactory sınıfının get metotları adaptörlerin üstsınıflarına ilişkin adresler geri döndürmektedir. Çünkü üst sınıfın adresini taşıyabilen bir işaretçi o sınıftan türeyen tüm alt sınıflara da işaret edebilir.

Soyut Fabrika (*Abstract Factory*) (GoF)

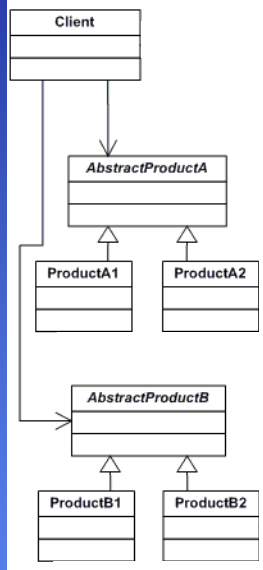
Bazı durumlarda birbirleriyle ilişkili birden fazla nesne yaratmak gerekir.

Örneğin yandaki şekilde gösterildiği gibi belli koşullarda ProductA1 sınıfından nesne ile ProductB1 sınıfından nesne birlikte kullanılacaktır.

Benzer şekilde de ProductA2 sınıfından nesne ile ProductB2 sınıfından nesne birlikte kullanılacaktır.

Bu durumda birbirleriyle ilişkili nesneleri yaratan farklı fabrika sınıfları oluşturulur.

Bu fabrika sınıfları ortak bir soyut fabrikadan türetilir.



3.Tekil Nesne (*Singleton*) (GoF)

Fabrika nesnesi yeni bir problem oluşturur: Fabrika nesnesini kim yaratacak ve bu nesneye nasıl erişilecek?

İstenenler:

- Fabrika sınıfından sadece tek bir nesne yaratılmalı
- Programın gerekli tüm yerlerinden bu nesneye erişilebilmeli.

Problem: Bir sınıftan sadece tek bir nesne yaratılması (tekil nesne) isteniyor. Bu nesne diğer nesnelere global tek bir erişim noktası sağlayacak.

Çözüm: Sınıfın içine tekil nesneyi yaratıp adresini döndüren statik bir metot yerleştirin.

Hatırlatma: Statik metotlar, üyesi oldukları sınıftan henüz bir nesne yaratılmadan önce de çağırılabilirler.

Örnek sistemde, ServicesFactory sınıfına, bu tipten nesnelere işaret edebilen statik bir nitelik üyesi, ve nesne yaratan statik bir metot eklenir.

Statik metot (getInstance) çağırıldığında eğer daha önceden bir fabrika nesnesi yaratılmamışsa nesne yaratır ve adresini geri gönderir.

Eğer daha önceden nesne yaratılmışsa yenisi yaratılmaz var olan nesnenin adresi gönderilir.