

**Hacettepe Üniversitesi**

**Bilgisayar Mühendisliği  
Bölümü**

**VERİ ERİŞİM NESNESİ**

Aslı Berivan Altuntaş  
20121923

# **İÇİNDEKİLER**

## **İçerik**

## **Problem**

## **Cözüm**

## **Veri Tabanına VEN İle Erişmek**

## **Yapı**

## **Katılımcılar ve Sorumlulukları**

### **İş Nesnesi**

### **Veri Erişim Nesnesi**

### **Veri Kaynağı**

### **Aktarım Nesnesi**

## **Yöntemler**

### **Otomatik VEN Kodu Üretim Yöntemi**

### **Veri Erişim Nesneleri için Fabrika Yöntemi**

## **Sonuçlar**

## **Gerçekleştirimler**

### **Veri Erişim Nesnesi Gerçekleştirimi**

### **Veri Erişim Nesneleri için Fabrika Yöntemi Gerçekleştirimi**

#### **Fabrika Yöntemi Tasarım Örüntüsü Kullanarak Fabrika Yönetiminin Gerçekleştirimi**

#### **Soyut Fabrika Tasarım Örüntüsü Kullanarak Fabrika Yönteminin Gerçekleştirimi**

## **Örnek Kodlar**

## **Kaynaklar**

## **İçerik:**

Veriye erişim verinin kaynağına bağlı olarak değişir . Veri tabanı gibi kalıcı depolara erişim, deponun ilişkisel veri tabanı ya da nesneye dayalı veri tabanı olmasına göre çeşitlilik gösterir ve gerçekleştirimi buna göre yapılır.

## **Problem:**

Java 2 Platformunda, (J2EE) uygulamaları belirli noktalarda kalıcı veriyi kullanmaya ihtiyaç duyarlar.Çoğu uygulamalar için kalıcı depolama işlemleri farklı mekanizmalarla gerçekleştirilir ve kalıcı veri depolarına erişim için belirgin farkları olan Uygulama Programlama Arayüzleri (UPA ) kullanılır. Bazı uygulamalar ise, ayrı sistemler üzerindeki verilere erişmek ihtiyacı duyabilirler.

Genellikle, uygulamalar kalıcı veriyi temsil eden varlık çekirdekleri gibi paylaşımlı, dağıtılmış bileşenleri kullanırlar.Bir uygulamada kalıcılığı sağlamak için gerekli SQL deyimleri varlık çekirdeği kodu içinde yazılıyorsa yani varlık çekirdeği açıkça kalıcı depoya erişebiliyorsa , çekirdek tabanlı kalıcılık yöntemi kullanılıyor demektir.Eğer uygulama varlık çekirdeklerini koza tabanlı kalıcılık yöntemi ile kullanıyorsa, birim işlem ve kalıcılık ayrıntıları koza tarafından ele alınır.Bu yöntemde,veri tabanı işlemleri varlık çekirdeği kodu içinde kodlanmak yerine yalnızca yaygınlaştırma belirtim kütüğünde gerekli belirtiler yapılır.Veritabanı ile ilgili işlemlerin sorumluluğu ve yönetimi işletim kozasına devredilmiş demektir.

İş uygulamaları genellikle ilişkisel ya da nesneye dayalı veri tabanlarına erişime ihtiyaç duyarlar ve Java platformu veriye erişim için farklı bazı teknikler sunar.Bu tekniklerden en eski ve iyi hazırlanmış teknik SQL sorgularını çalıştıran ve sonuçları getirmeyi sağlayanJava Veritabanı Bağlanabilirliği ( JVTB ) UPA' sını kullanmaktır.Bu UPA veriye erişim ve uygulamanın durumunun kalıcılığını sağlamak isteyen geliştiriciye bu imkanı sağlamasına rağmen, taşınması zor bir UPA'dır.

Uygulamalar, İlişkisel Veritabanı Yönetim Sistemi ( İVTYS )üzerinde yer alan verilere erişim için Java Veritabanı Bağlanabilirliği ( JVTB ) UPA' larını kullanabilirler JVTB UPA'sı, ilişkisel veritabanı gibi kalıcı bir depodaki verilere standart erişimi ve verilerin işlenmesini sağlar. JVTB UPA, J2EE uygulamalarının İVTYS tablolarına erişimi sağlayan SQL deyimlerini kullanabilmesine olanak verir. Fakat, İVTYS ortamında bile SQL deyimlerinin söz dizim ve biçim özellikleri özel veritabanı ürünlerine bağlı olarak değişebilir.

Farklı kalıcı veri depoları türleri için veriye erişimde daha büyük değişimler söz konusudur. UPA'larla desteklenen veriye erişim mekanizmaları ve özellikleri verinin kalıcılığının sağlanarak depolandığı yerlerin türlerine göre ( İVTYS, nesneye dayalı veri tabanı düz kütükler , vb...) değişiklik gösterirler. Tamamen farklı sistemlerdeki verilere erişmeye ihtiyaç duyan uygulamaların önceden geçerliliği kabul edilen UPA' ları kullanmaları gerekir. Bu şekilde tamamen farklı olan veri kaynaklar uygulamalara kimlik sormayı önerirler ve uygulama koduyla veri erişim kodu arasında doğrudan bir bağımlılık oluştururlar. İş bileşenleri olan , varlık çekirdekleri, oturum çekirdekleri ve sunucu yordamcığı ,JSP gibi sunum bileşenleri veriye erişim yapmak istediklerinde bağımlılığı elde etmek ve veri kaynağını işlemek üzere uygun UPA' ları kullanabilirler. Fakat bağımlılığını ve veriye erişimi kodunu aynı bileşen içinde birleştirmek bileşenler ile veri kaynağı gerçekleştirimini sıkı bir bağlaşım içine sokmaya sebep olur. Bu şekilde bir bağımlılık yaratmak , bileşeni gerçekleştirmeyi zorlaştırır ve uygulamanın üzerinde çalıştığı veri kaynağının türünü değiştirmek, uygulamayı farklı bir veri kaynağı üzerinde çalıştırmak güçleşir. Veri kaynağı değiştiği zaman yeni veri türünü işleyebilmesi için bileşenlerin de değiştirilmesi gerekir.

Veri kaynaklarının belirli özelliklerine bağlı olan kod ,iş mantığı ile veri erişim mantığını birbirine bağlar. Bu da uygulamaların veri kaynaklarının yer değiştirmesini veya değiştirilmesini zorlaştırır.

## **Çözüm:**

Veri kaynaklarına bütün erişimleri ayırmak ve sarmak için Veri Erişim Nesnesi ( VEN ) kullanılır. VEN, veriyi elde etmek ve depolamak için veri kaynağına bağlantıyı sağlar. VEN, üzerinde çalışılacak verinin bulunduğu veri kaynağına erişim mekanizmasını gerçekleştirir. Veri kaynakları farklı türlerde olabilir. İVTYS gibi kalıcı bir depo , işten işe ( *business-to-business* - B2B- ) bütünleşen sistemleri gibi dış hizmet sistemlerinden biri ya da Hafif Dizin Erişim İletişim Kuralları ( *Lightweight Directory Access Protocol* –LDAP- ) veri tabanı gibi havuz...

VEN' e bağlı olan iş bileşeni, VEN tarafından istemcileri için sunulan basit bir arayüzü kullanırlar. VEN, veri kaynağının gerçekleştirimin ayrıntılarını tamamen kullanıcılarında gizler. Çünkü, istemcilerine sunduğu arayüz altta bulunan veri kaynağı gerçekleştirimi değiştiginde değişmez. VEN, istemci ya da iş bileşenlerini etkilemeden , farklı depolama tasarımlarına uyarlanabilir. Aslında , VEN iş bileşenleri ile veri kaynağı arasında uyarlayıcı görevi yapar.

VEN tasarım örüntüsü , veri kaynağının istemci arayüzünü veri kaynağının veriye erişim mekanizmasından ayırır. Belir özellikleri olan bir veri kaynağının erişim UPA'sını genel özellikleri olan istemci arayüzüne uyarlar. VEN tasarım örüntüsü, veri erişim mekanizmalarının veriyi kullanan kod kesiminden bağımsız olarak değişebilmesine izin verir.

VEN, nesnenin kalıcılığını sağlamak ve veri erişim mantığını kendine has özellikleri olan kalıcılık yöntemleri veya UPA'dan ayıran bir teknik sağlar. VEN yaklaşımı uygulamanın kalıcılık sağlama mekanizmasının zamanla değiştirilmesine esneklik sağlar. Örneğin, varlık çekirdekleri kullanmak yerine doğrudan oturum çekirdeklerine JVTB çağrılarını yapma yöntemini kullanmak uygulamaya performans yararı sağlayabilir. VEN katmanı kullanılmadan bu geçiş var olan kod için pahalı bir yeniden yapılandırılma gerektirir.

VEN , sadece bir nesne ile bir ilişkisel tablo arasında basit bir eşleme yapmaz, aynı zamanda karmaşık sorguların gerçekleştirilmesine izin verir ve depolanmış yordamlarla veri tabanı görünümünün Java veri yapıları içinde eşlenmesine izin verir.

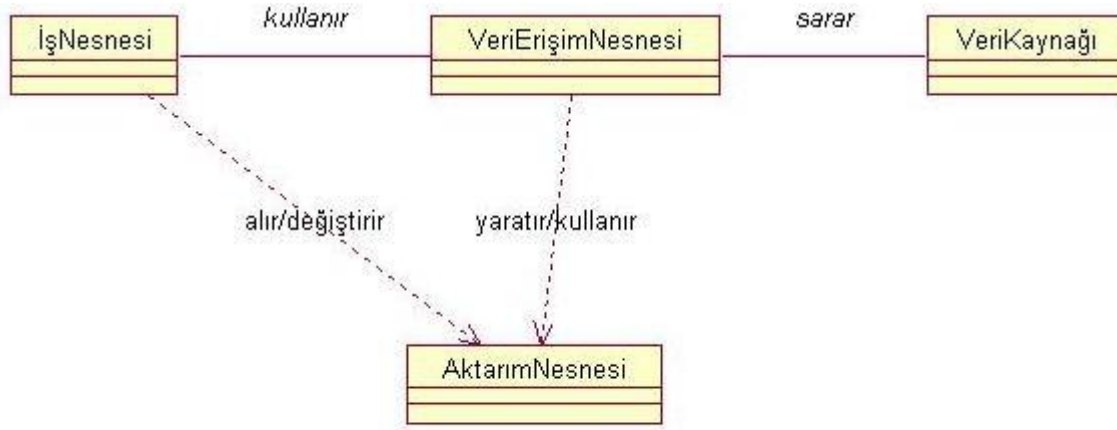
## **Veri Tabanına VEN ile Erişmek:**

VEN sınıfı kaynağın UPA'sı ile iş mantığını bağlamadan özel bir veri kaynağına erişimi sağlar. Örnek olarak verilen bir uygulamada, uygulama sınıfları bir VEN arayüzü ( KatalogVEN ) kullanarak kategorilere, ürünlere ve öğelere erişirler. KatalogVEN'nin farklı veri erişim mekanizmaları için yeniden gerçekleştirimi , onu kullanan sınıflar üzerinde küçük bir etki yaratabilir , ya da hiçbir etki de yaratmayabilir. Çünkü, sadece gerçekleştirim değişecektir. KatalogVEN'nin her muhtemel alternatif gerçekleştirimi , KatalogVEN'i kullanan her sınıfa aynı UPA sunuluyorken , katalog içindeki öğeler için kendi yoluyla verilere erişecektir.

Örnek Kodlar Bölümündeki Örnek.1 , bir uygulamanın veri kaynağına erişim mekanizmasından iş mantığını ayırmak için VEN tasarım örüntüsünü nasıl kullandığını örneklemektedir.

## **Yapı:**

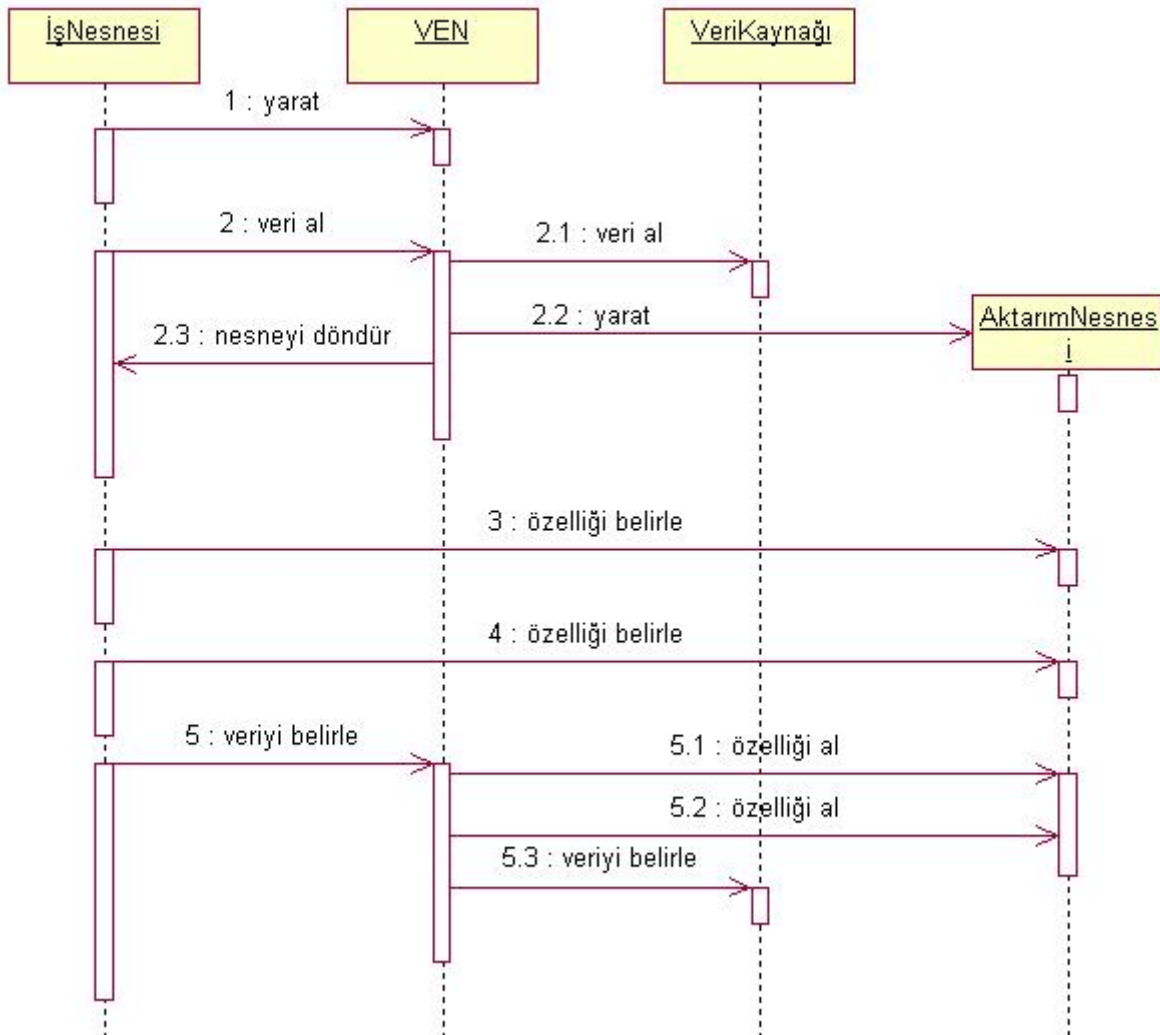
Şekil.1 VEN tasarım örüntüsü için kurulan ilişkileri temsil eden sınıf çizgesini göstermektedir.



Şekil.1 Veri Erişim Nesnesi

### Katılımcılar ve Sorumlulukları :

Şekil.2 bu tasarım örüntüsündeki değişik katılımcılar arasındaki etkileşim gösteren ardıl etkileşim çizgesini göstermektedir.



## Şekil.2 Veri Erişim Nesnesi Ardıl Etkileşim Çizgesi

### **İş Nesnesi:**

İş nesnesi veri istemcisini temsil eder. Veriyi elde etmek ve saklamak amacıyla veri kaynağına erişmek isteyen nesnedir. Bir iş nesnesi bir oturum çekirdeği, bir varlık çekirdeği ya da başka bir Java nesnesi, bir sunucu yordamcığı ya da veri kaynağına erişen yardımcı bir çekirdek olarak gerçekleştirilebilir.

### **Veri Erişim Nesnesi:**

VEN, bu tasarım örüntüsünün başlıca nesnesidir. VEN, alt kesimde yatan veri erişim gerçekleştirimini iş nesnesinin veri kaynağına erişiminin açık ve net olması için ayırır. Ayrıca iş nesnesi VEN'e veri yükleme ve depolama işlemleri yetkisini aktarır.

### **Veri Kaynağı:**

Veri kaynağı, veri kaynağı gerçekleştirimini temsil eder. Veri kaynağı, bir ilişkisel veri tabanı, nesneye dayalı bir veri tabanı, bir XML veri havuzu, düz kütük sistemi ve bunun gibi kaynaklar olabilir. Veri kaynağı ayrıca, başka bir sistem ( büyük bilgisayar ), hizmet ( B2B hizmeti ) ya da bir çeşit veri havuzu ( LDAP ) da olabilir.

### **Aktarım Nesnesi:**

Transfer nesnesi, veri taşıyıcı olarak kullanılır. VEN, transfer nesnesini istemciye veri döndürmesi için kullanabilir. VEN, ayrıca veri kaynağındaki verileri günlemek için aktarım nesnesi içinde istemciden gelen veriyi alabilir.

### **Yöntemler:**

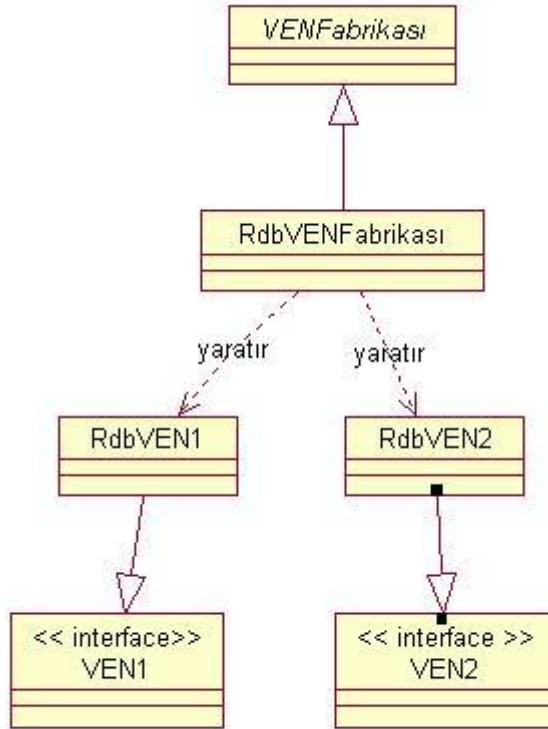
#### **Otomatik VEN Kodu Üretim Yöntemi :**

Her iş nesnesi belirli bir VEN ile uyduğu için, iş nesnesi, VEN ve altta yatan gerçekleştirimler ( ilişkisel bir veri tabanındaki tablolar gibi ) arasında bir ilişki kurulabilir. Bu ilişki bir kez kurulduktan sonra, basit bir uygulama yazmak mümkündür. Bu uygulama, belirli kod üretimi yapan bir yazılımdır. Bu yazılım ile bir uygulama tarafından istenilen tüm VEN, için kod üretilebilir. VEN'i üreten araveri geliştirici-tanımlı belirtim kütüğünden gelebilir. Alternatif olarak, kod üretici otomatik olarak veri tabanına erişim için gerekli VEN'leri sağlar. Eğer VEN için gereksinimler karmaşık ise, RDBMS veri tabanları için nesneden ilişkiye eşleme yapan üçüncü kısım araçları kullanmak düşünülebilir. Bu araçlar, iş nesnelere ve kalıcı depo nesnelere eşleştiren ve aracı VEN'leri tanımlayan tipik Grafik Kullanıcı Arayüzü araçlarını içerir. Ayrıca bu araçlar, eşleme tamamlandığında kodu bir kez üretirler,

ve sonuçları ön belleğe alma, sorguyu ön belleğe alma, uygulama sunucusu ile bütünleştirme, diğer üçüncü kısım ürünlerle bütünleştirme gibi sonradan katılan türdeki özellikleri sağlayabilirler.

### Veri Erişim Nesneleri için Fabrika Yöntemi:

VEN tasarım örüntüsü, Soyut Fabrika ( *Abstract Factory* ) ve Fabrika Yöntemi ( *Factory Method* ) tasarım örüntüleri ile birlikte kullanılarak yüksek derecede esneklik kazanabilir. Alt kesimdeki veri deposunun bir gerçekleştiriminden başka bir gerçekleştirimine geçişte bir değişiklik göstermesi söz konusu değilse , bu yöntem Fabrika Yöntemi tasarım örüntüsü kullanılarak uygulamanın ihtiyacı kadar VEN sağlanmasıyla gerçekleştirilmiş olur. Bu durum için geçerli olan sınıf çizgesi Şekil.3' de görülmektedir.

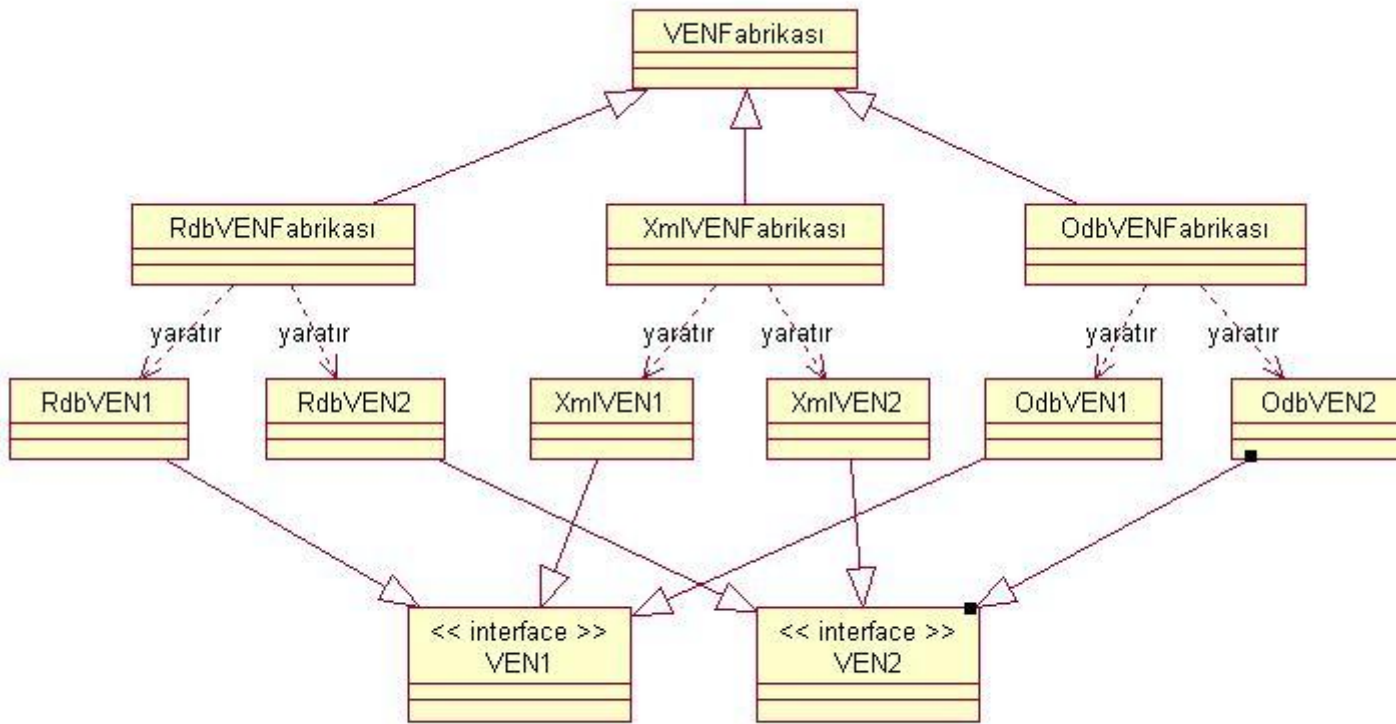


**Şekil.3 Fabrika Yöntemi Tasarım Örüntüsü Kullanılan VEN Fabrika Yöntemi**

Alt kesimdeki veri deposunun bir gerçekleştiriminden başka bir gerçekleştirimine geçişte bir değişiklik göstermesi söz konusu ise, bu yöntem Soyut Fabrika tasarım örüntüsü kullanılarak gerçekleştirilir. Bu durumda , bu yöntem değişik türde somut VEN fabrikaları üreten VEN fabrika nesnesi ( Soyut Fabrika nesnesi ) sağlar. Bu VEN fabrikalarının her biri , farklı türde kalıcı depolama gerçekleştirimini destekler. Belirli bir gerçekleştirim için somut

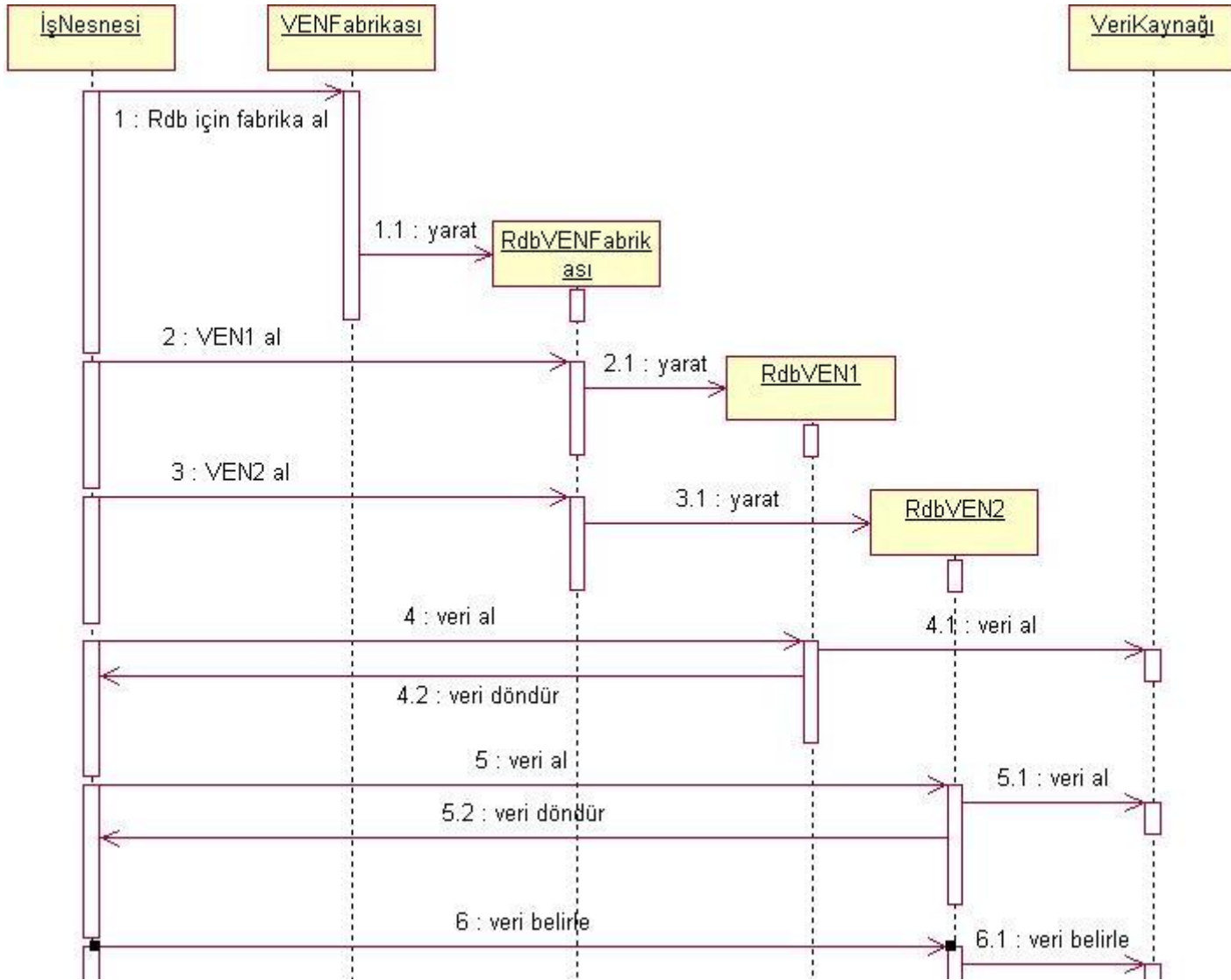
birVEN fabrikası bir kez elde edilir. Ve bu gerçekleştirim tarafından desteklenen ve gerçekleştirilen VEN'leri üretmek için elde edilmiş olan VEN fabrikası kullanılır. Bu yöntem için çizilen sınıf çizgesi Şekil.4'de yer almaktadır.

Temel VEN fabrikası soyut bir sınıftır. Bu sınıf farklı somut VEN fabrikaları tarafından saklama gerçekleştirimi özel erişim sağlamak amacıyla kalıtılır ve gerçekleştirilir. İstemci, RdbVENFabrika gibi bir somut VEN fabrika gerçekleştirimini elde edebilir ve onu belirli depolama gerçekleştirimleriyle çalışan somut VEN'ler elde etmek üzere kullanır. Örneğin, veri istemcisi RdbVENFabrika elde eder ve onu RdbMüşteriVEN, RdbHesapVEN gibi belirli VEN'leri almak için kullanır. VEN'ler VEN1 ve VEN2 ile gösterilen ve VEN'nin desteklediği iş nesnesi için gerekli gereksinimlerini belirli bir biçimde tanımlayan temel sınıfları genişletirler ve gerçekleştirimini yaparlar. Her somut VEN veri kaynağına bağlantı kurmaktan ve desteklediği iş nesnesi için verileri elde etmekten ve işlemekten sorumludur. VEN tasarım örüntüsü ve yöntemleri için örnek bir gerçekleştirim Örnek Kodlar kısmında yer almaktadır.



**Şekil.4 Soyut Fabrika Tasarım Örüntüsü Kullanılan VEN Fabrika Yöntemi**

Şekil.5'deki ardıl etkileşim çizgesi etkileşimleri göstermektedir.



**Şekil.5 Soyut Fabrika Tasarım Örüntüsü Kullanılan VEN Fabrika Yöntemi İçin Ardıl Etkileşim Çizgesi**

## Sonuçlar:

### Şeffaflık Sağlıyor

İş nesnelere veri kaynağının gerçekleştiriminin belirli ayrıntılarını bilmesine gerek kalmadan veri kaynağını kullanabilir. Erişim gerçekleştirim ayrıntıları VEN içinde gizlendiği için şeffaftır.

### Taşımayı Kolaylaştırır

Bir VEN katmanı, uygulamanın farklı veri tabanı gerçekleştirmelerine daha kolay bir şekilde taşınmasını sağlar. İş nesnelere alt kesimdeki veri gerçekleştirimi hakkında bir bilgiye

sahip değildir. Bu yüzden , taşıma sadece VEN katmanındaki değişiklikleri içerir. Üstelik, fabrika yöntemi çalıştırılırsa , alt kesimdeki her depolama gerçekleştirimi için bir somut fabrika gerçekleştirimi sağlamak mümkündür. Bu durumda farklı depolama gerçekleştirimlerine taşıma uygulamaya yeni fabrika gerçekleştirimi sağlamak anlamına gelir.

#### İş Nesnelerindeki Kod Karmaşıklığını Azaltır

VEN bütün veri erişim karmaşıklıklarını yönettiği için , iş nesnesi içindeki ve VEN'yi kullanan diğer veri istemcilerindeki kodu basitleştirir. SQL deyimleri gibi gerçekleştirim ilişkili bütün kodlar iş nesnesi içinde yer almaz ,VEN içinde yer alır. Bu kodun okunabilirliğini ve üretkenliği geliştirir .

#### Bütün Veri Erişimini Ayrı Bir Katman İçinde Merkezileştirir

Ayrı bir veri erişim katmanı uygulamanın geri kalanını veri erişim gerçekleştiriminden ayıran bir katman olarak görülebilir. Çünkü, bütün veri erişim işlemleri için yetki VEN'e aktarılmıştır.

#### Koza Tabanlı Kalıcılık Yöntemi İçin Kullanışlı Değildir

Koza tabanlı kalıcılık yönteminde EJB kozası varlık çekirdeklerini yönettiği için, bütün kalıcı depolara erişim hizmetini otomatik olarak verir. Koza tabanlı kalıcılık yöntemini kullanan uygulamalarda VEN katmanını kullanmaya gerek yoktur, çünkü uygulama sunucusu bu işlevi açık olarak sağlamaktadır. Fakat varlık çekirdekleri için koza tabanlı kalıcılık yöntemi ile oturum çekirdekleri , sunucu yordamcıkları için çekirdek tabanlı kalıcılık yönteminin birleştirilerek kullanımı gerektiğinde VEN' ler halen kullanışlıdır.

#### Fazladan Bir Katman Eklenir

VEN veri istemcisi ve veri kaynağı arasında ek bir katman oluşturur. Fakat, bu yaklaşım seçilerek elde edilen fayda ek olarak verilen çabanın karşılığını verir.

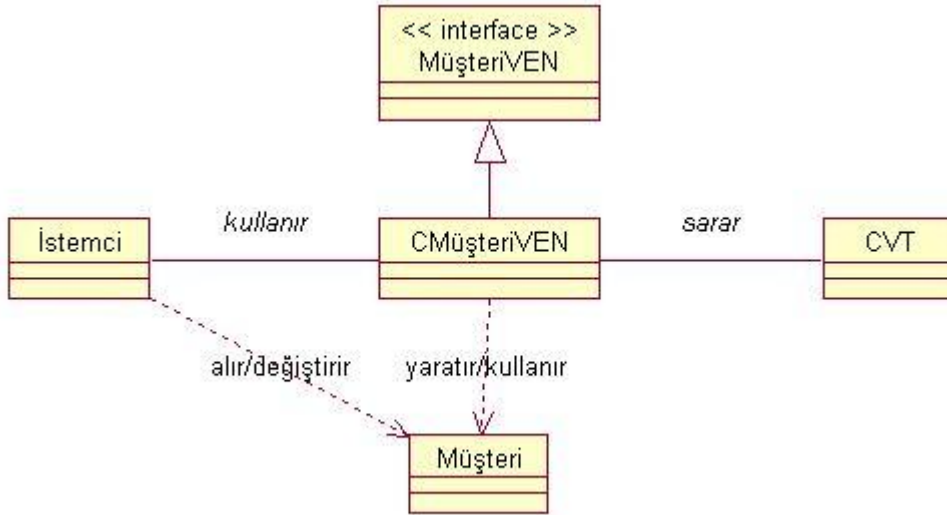
#### Sınıf Hiyerarşi Tasarımı Gerektirir

Fabrika yöntemi kullanıldığı zaman , somut fabrikaların hiyerarşisi ve bu somut fabrikalar tarafından üretilen somut ürünlerin hiyerarşisi tasarlanmalı ve gerçekleştirilmelidir. Verilen bu ek çaba eğer esnekliği sağlamak garanti ediliyorsa düşünülmelidir. Tasarımın karmaşıklılığını artırır. Fakat , önce Fabrika Yöntemi tasarım örüntüsü kullanılarak uygulanan Fabrika yöntemi gerçekleştirimi seçilebilir ve daha sonradan gerekirse Soyut Fabrika tasarım örüntüsü kullanılarak uygulanan Fabrika yöntemine geçiş yapılabilir.

## Gerçekleştirmeler :

### Veri Erişim Nesnesi Gerçekleştirimi:

Müşteri bilgisini temsil eden kalıcı nesne için bir VEN örnek kodu Örnek.5’de görülmektedir. findCustomer() yordamı çağrıldığında CMüşteriVEN Müşteri Aktarım Nesnesini yaratır.VEN’i kullanmak için örnek bir kod Örnek.7’de görülmektedir. Örneğin sınıf çizgesi Şekil.6’da gösterilmektedir.



Şekil.6 VEN Tasarım Örüntüsünün Gerçekleştirimi

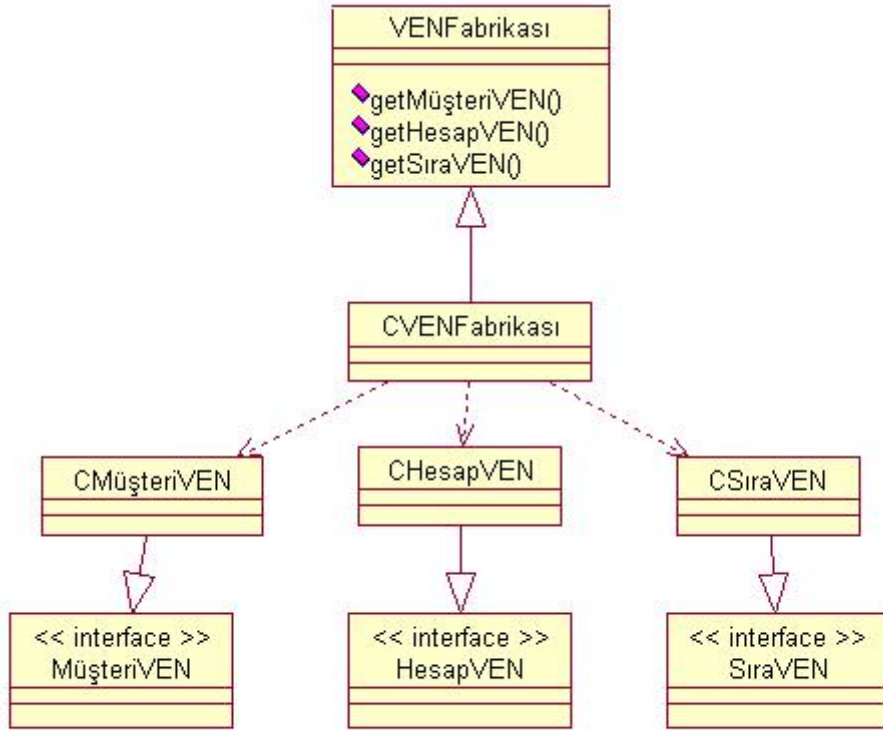
### Veri Erişim Nesneleri için Fabrika Yöntemi Gerçekleştirimi :

#### **Fabrika Yöntemi Tasarım Örüntüsü Kullanarak Fabrika Yönetiminin Gerçekleştirimi**

Tek bir veri tabanı gerçekleştirimi için VEN fabrikasının birden fazla VEN’i ürettiği bir ortamda Fabrika Yöntemini gerçekleştirdiğimiz bir örneği düşünelim. Fabrika MüşteriVEN, HesapVEN,SıraVEN gibi VEN’leri üretir.

VEN fabrikası ( CVENFabrikası ) için örnek kod Örnek.3’de görülmektedir.

Örneğin sınıf çizgesi Şekil.7’de görülmektedir.

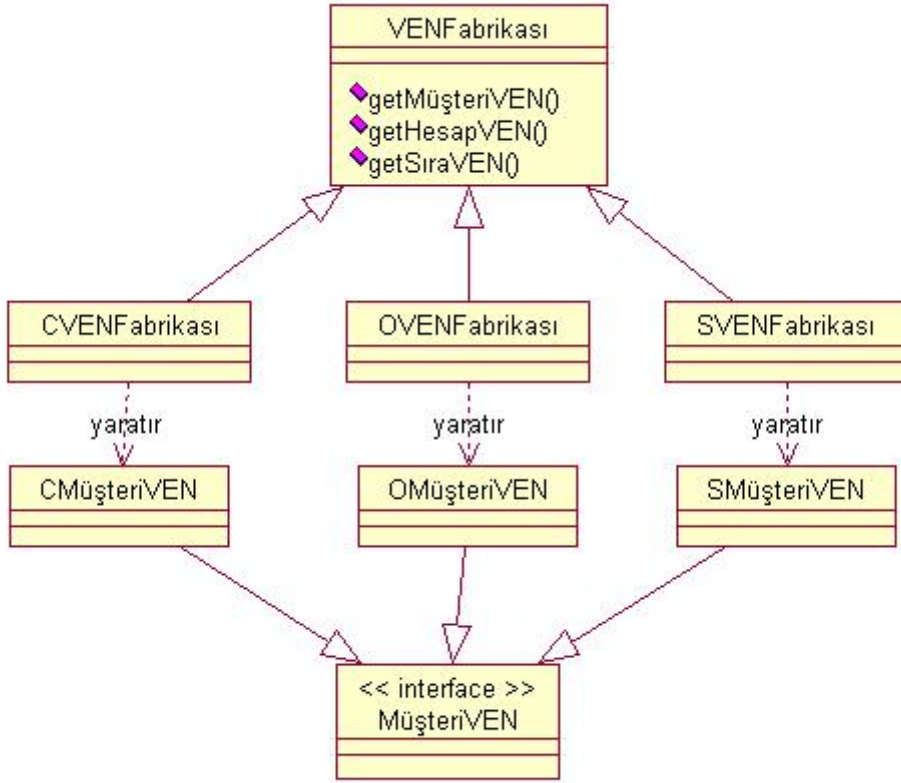


**Şekil.7 Fabrika Yöntemi Tasarım Örüntüsü Kullanarak VEN için Fabrika Yönteminin Gerçekleştirimi**

### **Soyut Fabrika Tasarım Örüntüsü Kullanarak Fabrika Yönteminin Gerçekleştirimi**

Üç farklı veri tabanı için Fabrika Yöntemini gerçekleştirdiğimiz bir örneği düşünelim. Bu durumda Soyut Fabrika tasarım örüntüsü kullanılır. Örneğin sınıf çizgesi Şekil.8’de görülmektedir.

Soyut VENFabrika sınıfı için bir alıntı kod parçası Örnek.2’de görülmektedir. Bu fabrika MüsteriVEN, HesapVEN, SıraVEN gibi VEN’leri üretiyor. Burada Soyut Fabrika tarafından üretilen fabrikalar içindeki Fabrika Yöntemi gerçekleştirimleri kullanılmaktadır.



**Şekil.8 Soyut Fabrika Tasarım Örüntüsü Kullanarak Fabrika Yönteminin Gerçekleştirimi**

## Örnek Kodlar:

### Örnek.1

CatalogDAO arayüzü VEN UPA'sını tanımlar. Arayüzdeki yöntemler belirgin veri erişim mekanizmalarına başvuru yapmazlar.Örneğin , hiçbiri bir SQL sorgusu belirtmezler, ve sadece CatalogDAOSysException türündeki aykırı durumları fırlatırlar.Aykırı durum fırlatmak dahil mekanizmaya özgü bilgileri VEN arayüzü içinde bulundurmaktan kaçınmak, gerçekleştirim ayrıntılarını gizlemenin temelini oluşturmaktadır.

```

public interface CatalogDAO {
    public Category getCategory(String categoryID, Locale l)
        throws CatalogDAOSysException;
    public Page getCategories(int start, int count, Locale l)
        throws CatalogDAOSysException;
    public Product getProduct(String productID, Locale l)
        throws CatalogDAOSysException;
    public Page getProducts(String categoryID, int start,int count,Locale l)
        throws CatalogDAOSysException;
    public Item getItem(String itemID, Locale l)
        throws CatalogDAOSysException;
    public Page.getItems(String productID, int start, int size, Locale l)
        throws CatalogDAOSysException;
    public Page searchItems(String query, int start, int size, Locale l)
        throws CatalogDAOSysException;
}

```

Örnek olarak verilen alıntı kod parçasında CloudscapeCatalogDAO sınıfı Cloudscape ilişkisel veri tabanı için CatalogDAO arayüzünü gerçekleştirir .

```

public class CloudscapeCatalogDAO implements CatalogDAO {
    ...
    public static String GET_CATEGORY_STATEMENT
        = "select name, descn "
        + " from (category a join category_details b on a.catid=b.catid) "
        + " where locale = ? and a.catid = ?";
    ...
    public Category getCategory(String categoryID, Locale l)
        throws CatalogDAOSysException {
        Connection c = null;
        PreparedStatement ps = null;
        ResultSet rs = null;
        Category ret = null;

        try {
            c = getDataSource().getConnection();
            ps = c.prepareStatement(GET_CATEGORY_STATEMENT,
                ResultSet.TYPE_SCROLL_INSENSITIVE,
                ResultSet.CONCUR_READ_ONLY);
            ps.setString(1, l.toString());
            ps.setString(2, categoryID);
            rs = ps.executeQuery();
            if (rs.first()) {
                ret = new Category(categoryID ,rs.getString(1),
                    rs.getString(2));
            }
            rs.close();
            ps.close();
            c.close();
            return ret;
        } catch (SQLException se) {
            throw new CatalogDAOSysException("SQLException: " +
                se.getMessage());
        }
    }
}

```

```
    ...  
    }  
Örnek.2
```

## Soyut VEN Fabrika Sınıfı

```
// Abstract class DAO Factory  
public abstract class DAOFactory {  
  
    // List of DAO types supported by the factory  
    public static final int CLOUDSCAPE = 1;  
    public static final int ORACLE = 2;  
    public static final int SYBASE = 3;  
    ...  
  
    // There will be a method for each DAO that can be  
    // created. The concrete factories will have to  
    // implement these methods.  
    public abstract CustomerDAO getCustomerDAO();  
    public abstract AccountDAO getAccountDAO();  
    public abstract OrderDAO getOrderDAO();  
    ...  
  
    public static DAOFactory getDAOFactory(  
        int whichFactory) {  
  
        switch (whichFactory) {  
            case CLOUDSCAPE:  
                return new CloudscapeDAOFactory();  
            case ORACLE :  
                return new OracleDAOFactory();  
            case SYBASE :  
                return new SybaseDAOFactory();  
            ...  
            default :  
                return null;  
        }  
    }  
}
```

## Örnek.3

### Cloudscape İçin Somut VEN Fabrikası Gerçekleştirimi

Oracle VEN Fabrikası ve Sybase VEN Fabrikası için yapılacak gerçekleştirmeler bazı gerçekleştirim ayrıntıları dışında benzerdir.

```
// Cloudscape concrete DAO Factory implementation  
import java.sql.*;  
  
public class CloudscapeDAOFactory extends DAOFactory {  
    public static final String DRIVER=  
        "COM.cloudscape.core.RmiJdbcDriver";  
    public static final String DBURL=  
        "jdbc:cloudscape:rmi://localhost:1099/CoreJ2EEDB";  
  
    // method to create Cloudscape connections  
    public static Connection createConnection() {
```

```

    // Use DRIVER and DBURL to create a connection
    // Recommend connection pool implementation/usage
}
public CustomerDAO getCustomerDAO() {
    // CloudscapeCustomerDAO implements CustomerDAO
    return new CloudscapeCustomerDAO();
}
public AccountDAO getAccountDAO() {
    // CloudscapeAccountDAO implements AccountDAO
    return new CloudscapeAccountDAO();
}
public OrderDAO getOrderDAO() {
    // CloudscapeOrderDAO implements OrderDAO
    return new CloudscapeOrderDAO();
}
...
}

```

## Örnek.4

### Müşteri İçin Temel VEN Arayüzü

MüşteriVEN arayüzü, somut VEN gerçekleştirmeleri (CMüşteriVEN, OMüşteriVEN, SMüşteriVEN) tarafından gerçekleştirilen Müşteri kalıcı nesnesi için VEN yöntemlerini tanımlar. Hesap ve Sıra iş nesneleri için VEN yöntemleri tanımlayan HesapVEN ve SıraVEN arayüzleri benzer şekildedir.

```

// Interface that all CustomerDAOs must support
public interface CustomerDAO {
    public int insertCustomer(...);
    public boolean deleteCustomer(...);
    public Customer findCustomer(...);
    public boolean updateCustomer(...);
    public RowSet selectCustomersRS(...);
    public Collection selectCustomersTO(...);
    ...
}

```

## Örnek.5

### Müşteri İçin Cloudscape VEN Gerçekleştirimi

```

// CloudscapeCustomerDAO implementation of the
// CustomerDAO interface. This class can contain all
// Cloudscape specific code and SQL statements.
// The client is thus shielded from knowing
// these implementation details.

import java.sql.*;

public class CloudscapeCustomerDAO implements
    CustomerDAO {

    public CloudscapeCustomerDAO() {
        // initialization
    }
}

```

```

}

// The following methods can use
// CloudscapeDAOFactory.createConnection()
// to get a connection as required

public int insertCustomer(...) {
    // Implement insert customer here.
    // Return newly created customer number
    // or a -1 on error
}

public boolean deleteCustomer(...) {
    // Implement delete customer here
    // Return true on success, false on failure
}

public Customer findCustomer(...) {
    // Implement find a customer here using supplied
    // argument values as search criteria
    // Return a Transfer Object if found,
    // return null on error or if not found
}

public boolean updateCustomer(...) {
    // implement update record here using data
    // from the customerData Transfer Object
    // Return true on success, false on failure or
    // error
}

public RowSet selectCustomersRS(...) {
    // implement search customers here using the
    // supplied criteria.
    // Return a RowSet.
}

public Collection selectCustomersTO(...) {
    // implement search customers here using the
    // supplied criteria.
    // Alternatively, implement to return a Collection
    // of Transfer Objects.
}
...
}

```

## Örnek.6

### Müşteri Aktarım Nesnesi

```

public class Customer implements java.io.Serializable {
    // member variables
    int CustomerNumber;
    String name;
    String streetAddress;
    String city;
    ...

    // getter and setter methods...
    ...}

```

## Örnek.7

### VEN ve VEN Fabrikası Kullanımı İçin İstemci Kodu

Kod parçası VEN ve VEN Fabrikası kullanımını örnekler. Eğer gerçekleştirim Cloudscape ürününden başka bir ürüne değişirse , gerekli olan değişiklik farklı bir fabrika elde etmek için VEN Fabrikası'nın getDAOFactory() yönteminin çağrılmasıdır.

...

```
// create the required DAO Factory
DAOFactory cloudscapeFactory =
    DAOFactory.getDAOFactory(DAOFactory.DAOCLOUDSCAPE);

// Create a DAO
CustomerDAO custDAO =
    cloudscapeFactory.getCustomerDAO();

// create a new customer
int newCustNo = custDAO.insertCustomer(...);

// Find a customer object. Get the Transfer Object.
Customer cust = custDAO.findCustomer(...);

// modify the values in the Transfer Object.
cust.setAddress(...);
cust.setEmail(...);
// update the customer object using the DAO
custDAO.updateCustomer(cust);

// delete a customer object
custDAO.deleteCustomer(...);
// select all customers in the same city
Customer criteria=new Customer();
criteria.setCity("New York");
Collection customersList =
    custDAO.selectCustomersTO(criteria);
// returns customersList - collection of Customer
// Transfer Objects. iterate through this collection to
// get values.
```

...

## Kaynaklar :

<http://www.codefutures.com/products/firestorm/benefits/>

<http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html>

<http://java.sun.com/blueprints/patterns/DAO.html>

[http://eski.tbd.org.tr/ing\\_turk.html](http://eski.tbd.org.tr/ing_turk.html)

