HACETTEPE ÜNİVERSİTESİ BİLGİSAYAR

MÜHENDİSLİĞİ BÖLÜMÜ

HIBERNATE

HAZIRLAYANLAR

Ekip	İsim - Soyisim	E-Posta	Tarih
2. Ekip	Mehtap Kaya İsmail Özen	kayamehtap@gmail.com ismozen@gmail.com	21.01.2007
1. Ekip	Adem Öztürk		27.12.2004

Belgeye ilişkin eleştiri ve önerilerinizi lütfen Ar. Gör. Fatih M. GÜLEÇ'e bildirin.
fatihg@cs.hacettepe.edu.tr

1-Giriş

Kalıcı verilerin yönetim şekli bütün yazılım projeleri için önemli bir anahtar nokta olmuştur.Verinin kalıcılığının sağlanması verinin veri tabanında ya da herhangi bir kütükte tutulması demektir.Kalıcı verilerin veri tabanında tutulduğunu varsayarsak uygulamalarımız da ilkel sorgulama işlemlerini(create,insert, update,delete) SQL ve JDBC de el ile mi yazmalıyız yoksa bunlar otomatik olarak mı oluşturulmalı bu durumda her veritabanı yönetim sisteminin kendi sorgu diyalekti varken taşınabilirlik nasıl başarılacak gibi sorunlarla karşışmaktayız. Normalde Java'da kalıcılık verinin ilişkisel veri tabanına SQL ile kaydedilmesidir. Ama burada sorun bu işi nesneye dayalı uygulamalarda en başarılı şekilde nasıl yapacağımızdır.Bu sorunların cevapları tartışılırken, son zamanlarda Nesne İlişkisel Eşleme (ORM(Object/Relational Mapping)) büyük kabul gördü.Nesne İlişkisel Eşleme, İlişkisel Veri tabanı ile Nesneler arasında çevirme yapma ve bağlantı kurma işlemidir.Küçük nesnelerle bu işlemi yapmak kolay iken nesnelerin, verilerin ve birleştirmemiz gereken tabloların sayısı arttıkça, bu işleminin yapılması oldukça zor bir hal alır.Karmaşıklaşan nesne modeli ve ilişkileri karşımıza sorun olarak çıkar.

Hibernate bu nokta da bu işlemleri en başarılı şekilde yerine getirmemizde bize büyük kolaylık, rahatlık ve başarım sağlar. Hibernate Java'da kalıcı veri yönetimine bütün bir çözüm getiren bir projedir. Java için bir ORM / Object-Relational Mapping Kitaplığı yani bir Nesne/İlişkisel Eşleme aracıdır. Uygulamaların, ilişkisel veri tabanı ile etkileşimine aracılık eder. Basit bir Java nesnesini kalıcı hale gelmesini ve kaydedilmiş kalıcı nesneyi geri yüklememizi basit komutlarla sağlar. Buda geliştiricinin sadece iş mantığına odaklanmasını sağlar.Takip edilmesini gerektiren belli katı kuralları yoktur. Bu sayede hem yeni hem de var olan projelere herhangi bir değişiklik gerektirmeden başarılı bir şekilde uyum sağlar.

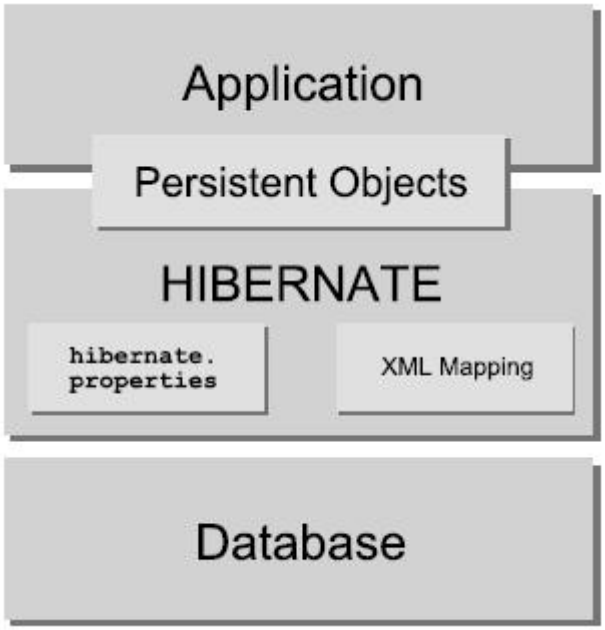
Java dünyasında ilişkisel veri tabanlarına ilk erişim yöntemi JDBC olmuştur. Fakat burada basit sorgulama komutlarını dahi elle yazmamız gerekiyordu. Bu da geliştiriciye büyük yük getiriyordu. Bu yüzden yazılan uygulama kodlarıda veri yapısına bağımlıydı. Yazdığımız nesneye dayalı kodlamaya rağmen veriye erişim nesnesel olmayan bir şekilde yapılıyordu. Nesneye dayalı programlama ile uğraşan geliştiriciler, veriye de yine nesneler ile ulaşmak ister. Bütün bu ihtiyaçları en iyi yerine getiren Hibernate oldu. Geliştiricisi Gavin King'e göre kalıcı nesneler, başka bir nesneden türetilmeyen basit Java nesneleri POJO(Plan Old Java Object) olmalıydı. Belirli katı standartları olmamalı ve güçlü bir sorgulama dilini desteklemeliydi. Ve bunların hepsi açık kaynak kodlu olmalıydı.

Hibernate,ayrıca HQL adında SQL'e benzeyen bir sorgulama dilini destekler. HQL'in tablolar yerine sınıflar üzerinde çalışan bir sorgulama dili olması nesneye yönelik yazılım ile uğraşan geliştiriciler için oldukça önemli bir getiridir

2-Mimari

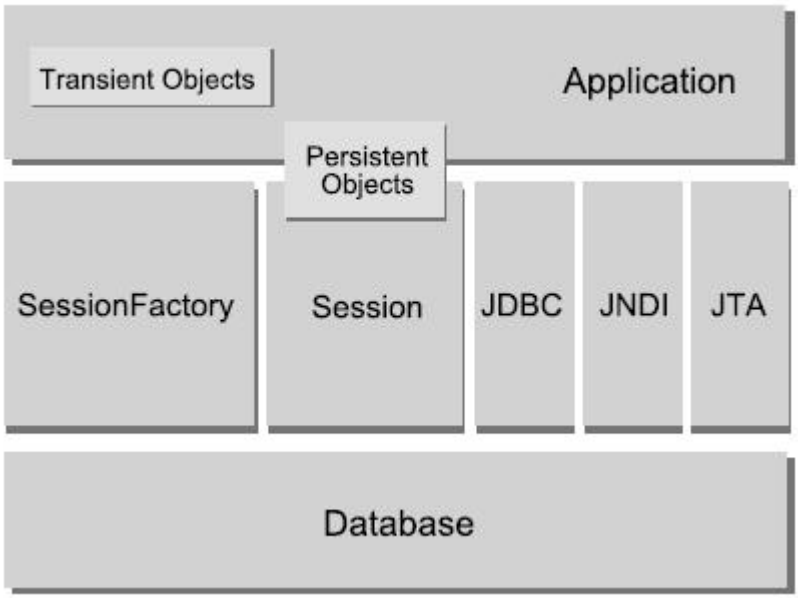
2.1-Genel Görünüm

Aşağıda ki diyagram da Hibernate'in uygulamaya veri kalıcılığı sağlamak için veri tabanı ve yapılandırma kütüklerini kullandığı görülmektedir.



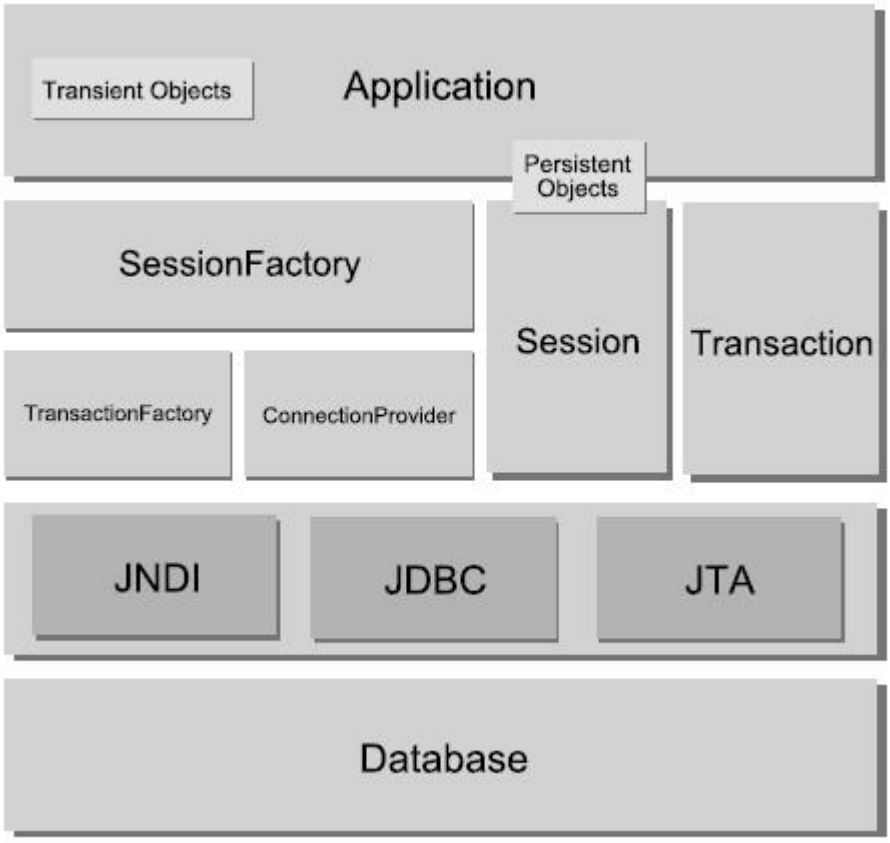
Üst düzey Hibernate Mimarisi

Daha detaylı mimari gösterimleri aşağıda verilmiştir:



Lite mimarisi

Bu mimaride uygulama JDBC bağlantılarını kendisi sağlar ve transactionları kendisi yönetir.



Full-cream mimarisi

Bu mimari uygulamayı altta yatan JDBC/JTA/JNDI katmanlarından soyutlar ve detayları Hibernate'e bırakır.

Bu mimari de geçen nesnelerin tanımları aşağıda verilmiştir.

SessionFactory :

SessionFactory derlenmiş basit bir veri tabanı adreslemelerinin tutulduğu alan olarak tanımlanabilir.

Session :

Uygulama ve kalıcı veriler arasında ki tek iş parçacıklı kısa süreli bir görüşmeyi temsil eder.

Persistent objects and collections:

İş methodları içerebilen kalıcı durumlu tek iş parçacıklı,kısa ömürlü nesnelerdir.Bu nesneler belli bir anda tek bir oturumla(Session nesnesiyle) ilişkili olup oturum sonlandığında nesneler serbest kalır ve herhangi bir uygulama katmanı tarafından kullanıma hazır hale gelirler.

Transient and detached objects and collections:

Bir oturumla(Session nesnesiyle) ilişkilendirilmemiş kalıcı sınıf nesneleridir.

Transaction:

Uygulama tarafından işin atomik birimlerini belirtmek amacıyla kullanılan tek iş parçacıklı,kısa ömürlü nesnelerdir.Bir Session belli koşullarda birden çok transactiona yayılabilmektedir.
Transactionlar uygulamayı JDBC, JTA ya da CORBA alt katmanlarından yalıtır.

ConnectionProvider :

JDBC bağlantılarının tutulduğu bir havuz ya da JDBC bağlantıları üreten bir fabrika olarak düşünülebilir. Uygulamayı altta yatan Datasource ve DriverManager katmanlarından soyutlar.

TransactionFactory :

Transactionlar için bir fabrika işlevi görür.

2.2. Nesne Durumları

Bir kalıcı sınıf nesnesi aşağıdaki 3 durumdan herhangi birinde bulunabilir.

Transient(geçici)
Nesne henüz hiçbir kalıcı içerikle ilişkilendirilmemiştir.

Persistent(kalıcı)
Nesne halen kalıcı bir içerikle ilişkilendirilmiş durumdadır.nesnenin birincil anahtarı yani veri tabanında ilişkilendirildiği bir satır mevcuttur.
Kimi önemli kalıcı içeriğin kimliğinin java kimliğine eşit olması hibernate tarafından garanti edilir.

Detached(ayrık)
Nesne,daha öncesinde kalıcı bir içerikle ilişkilendirilmişdir fakat bu içerik kapatılmıştır.Nesnenin kalıcı kimliği ve veri tabanında karşılık bulduğu bir satır mevcuttur.
Detached nesnelerin kalıcı kimlik ve java kimliği uyuşması hibernate tarafından agranti edilmez.

3-Yapılandırma

3.1. SessionFactory yaratmak

Bir sessionFactory nesnesi yaratabilmek için ilk önce uygulama yüklenirken eşleme(mapping) dosyalarının yerlerini ayarlamak amacıyla Configuration sınıfının bir kopyasını oluşturmamız gerekmektedir. Örnek kod hibernate'in başlamasını sağlayan kod bloğudur.

```
Configuration cfg = new Configuration();
cfg.addResource("hello/Message.hbm.xml");
cfg.setProperties( System.getProperties() );
SessionFactory sessions = cfg.buildSessionFactory();
```

Bu kod bloğu sayesinde SessionFactory yaratmış oluruz. Bu nesne tüm uygulama iş parçacıkları (thread) tarafından ortak bir şekilde kullanılır. Hibernate birden fazla SessionFactory elde edilmesine izin vermez.

Hibernate'te bu XML eşleme dosyaları .hbm.xml uzantılı olmak zorundadır ve bunlar her bir sınıf için hepsi bir XML eşleme dosyası yerine ayrı ayrı oluşturulmalıdır. Hibernate dokümantasyonu her kalıcı sınıf için oluşturulan bu XML eşleme dosyası aynı dizine konulmalıdır.

```
SessionFactory sessions = new Configuration()
.addClass(org.hibernate.auction.model.Item.class)
.addClass(org.hibernate.auction.model.Category.class)
.addClass(org.hibernate.auction.model.Bid.class)
.setProperties( System.getProperties() )
.buildSessionFactory();
```

Yukarıdaki kod bloğundaki addClass() metodu eşleme dosyalarının .hbm.xml uzantısı olduğunu ve kalıcı sınıfla aynı dizinde yer aldığını varsayar.

Tabiki hibernate ayarlarını yapmak sadece eşleme dosyalarının nerde olduğunu göstererek bitmiyor. Bunlarla birlikte veritabanı bağlantısının nasıl yapılacağını da ayarlamak gerekmektedir.

Belgeye ilişkin eleştiri ve önerilerinizi lütfen Ar. Gör. Fatih M. GÜLEÇ'e bildirin.
fatihg@cs.hacettepe.edu.tr

3.2.Ayarlar yapılırken XML tabanlı dosya kullanmak

Eğer Sessionfactory'i tam olarak ayarlamak istiyorsak XML ayar dosyası kullanabiliriz. Hibernate.properties sınıfının aksine hibernate.cfg.xml dosyası sadece ayarlama parametrelerini içermez. Bunun yanı sıra kalıcı sınıf eşleme dosyalarının nerde olduğu bilgiside ayarlayabilir. Birçok kullanıcı kod içerisinde hibernate ayarlarını yapmaktansa bu şekilde XML dosyaları oluşturarak halleder.

Aşağıdaki örnekte görüldüğü üzere veritabanı ile ilgili bilgilerde bu XML dosyasına yazılmıştır. Dolayısıyla kod içerisinde veritabanı işlemleri için herhangi bir ayar yapmamıza gerek kalmamıştır. Tüm ayarlar bu XML dosyası kullanılarak yapılacaktır. Yine kalıcı sınıfların eşleme dosyalarının da bulundukları yerler yine bu XML dosyasına eklenmiştir.

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
<session-factory>
<property
name="hibernate.connection.driver_class">org.postgresql.Driver</prop
erty>
<property
name="hibernate.connection.url">jdbc:postgresql://localhost/deneme</
property>
<property name="hibernate.connection.username">postgres</property>
<property name="hibernate.connection.password">123456</property>
<property name="hibernate.connection.pool_size">10</property>
<property name="show_sql">true</property>
<property
name="dialect">org.hibernate.dialect.PostgreSQLDialect</property>
<property
name="transaction.factory_class">org.hibernate.transaction.JDBCTrans
actionFactory</property>

<!-- Mapping files -->
<mapping resource="Objects/yemek.hbm.xml"/>
</session-factory>
```

4-Kalıcı Sınıflar

Kalıcı sınıflar yapılacak işe ait varlıklara karşılık gelirler.Kalıcılık,verinin herhangi bir veri saklama ortamında (veri tabanı,kütükler) saklanmasıyla sağlanır. Örneğin veri tabanında tuttuğumuz,bir yemek sipariş sisteminde ki yemek sınıfını kalıcı bir sınıftır. Kalıcı sınıfların tüm örneklemlerinin kalıcı olması beklenmez.

Hibernate kalıcı sınıfların Plain Old Java Objects (POJO) denilen çok sıkı gereksinimleri bulunmayan programlama modeline uymasını bekler.

```
package Objects;

public class Yemek {

	private int id; //belirleyici nitelik
	private String yemekadi;
	private int fiyati;

	public Yemek(){

	}
	public Yemek(int id, String yemekadi, int fiyati) {
		this.id = id;
		this.yemekadi = yemekadi;
		this.fiyati = fiyati;
	}
	public int getFiyati() {
		return fiyati;
	}
	public void setFiyati(int fiyati) {
		this.fiyati = fiyati;
	}
	public int getId() {
		return id;
	}
	public void setId(int id) {
		this.id = id;
	}
	public String getYemekadi() {
		return yemekadi;
	}
	public void setYemekadi(String yemekadi) {
		this.yemekadi = yemekadi;
	}

}
```

Kalıcı sınıfların özellikleri

- Kalıcı sınıflar javabean tarzında yazılmalıdır çünkü hibernate de verilerin Javabean tarzında yazılmış olduğunu varsayar.JavaBean tarzı sınıf demek tüm nitelikleri private olan, belirleyici niteliği sıralı(serialized) olan, niteliklerine ulaşmak için get/set methodları tanımlayan ve boş bir yapıcısı bulunan sınıf demektir.Bu özellikler aşağıda ayrıntılı oalrak açıklanmıştır.

- Tüm kalıcı sınıfların public olan bir önbelirli yapılandırıcısının olması gerekir.Bu Hibernate'in Constructor.newInstance() methodunu kullanmasını sağlar.

- Yemek sınıfının id isimli bir niteliği bulunmaktadır.Bu özellik veri tabanında ki yemek tablosunun birincil anahtarıyla adreslenir.Bu tür niteliklere belirleyici nitelik denir.Belirleyici niteliklerin varlığı zorunlu olmamakla birlikte belirleyici nitelik bildirimi yapan sınıfların fonksiyonelliği artar.Belirleyici niteliğe değer atama otomatik olarak yapılır bu aynı belirleyici nitelikle başka bir nesne yaratılmaya çalışılmasının önünü keser.

- Yemek sınıfı kalıcı methodlarına erişmek için getter/seter methodlar tanımlamıştır.İlişkisel şema ve sınıf nitelikleri arasında dolaylı bir iletişim kurma mantığında olan hibernate bunu get, set ve is methodlarıyla sağlar.

5-Nesne İlişkisel Eşleme

Nesne ilişkisel Eşleme ilişkisel veritabanı sistemlerinde bulunan nesneler ile uygulama tarafında bulunan nesnelerin ilişkilendirilmesi anlamına gelen bir terimdir**.** Nesne ilişkisel Eşleme araçlarının çok katmanlı mimarideki yeri veri erişim katmanı olarak isimlendirilebilen katmandır.

Nesne ilişkisel Eşleme araçları kendi alt yapılarında bulunan eşleme yöntemlerine göre illişkisel veritabanı yönetim sisteminde bulunan tablolar ile bunlara karşılık gelen uygulama nesneleri arasında veri aktarımını sağlar. Yani verileri çekip nesnelere yüklemek için geliştirici tarafından herhangi bir kod yazılmaz, tanımlanan adresleme yöntemine göre gereken sql sorgusu Nesne ilişkisel Eşleme aracı tarafından oluşturulur ve dönen veriler nesnelere yüklenir. Nesneleri saklanması, nesnelere ulaşılması, silinmesi, güncellenmesi gibi işlemler için alt yapıda tanımlanmış olan veritabanı yönetim sisteminin anlayacağı dilde sql sorgusunu oluşturulması ve işletilmesi tamamen Nesne ilişkisel Eşleme aracının işidir.

Veri tabanına erişim işlemlerini geliştiriciye bırakmanın bazı sakıncaları vardır.Aynı kod kesimi uygulamanın bir çok yerinde tekrarlanmakta olduğundan kodun güncellenebilirliği ve yeniden kullanımı oldukça azalmaktadır.Veri tabanı erişim mekanizmasının uygulama mantığından ayırıp ayrı bir katmanda ele alınması yazılımın güncelelnebilirliğini ve yeniden kulanılabilirliğini oldukça arttırmaktadır.Nesne İlişkise Eşleme mantığının sağladığı bir diğer avantaj ise veri tabanı bağlantıları,işlemleri için harcanan zamanı azaltmasıdır.

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
"-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-
3.0.dtd">

<hibernate-mapping package="Objects">
     <class name="Yemek" table="yemek">
          <id name="id" type="java.lang.Integer">
               <column name =”id”>
               <generator
          class=”increment”></generator>
          </id>
          <property name="yemekadi">
           <column name="yemekadi" />
          </property>
          <property name="fiyati">
          <column name="fiyati"/>
          </property>
     </class>
</hibernate-mapping>
</hibernate-mapping>
```

5.1 Nesne/ilişkisel model eşleme(ORM) kütüğü içeriği

5.1.1 Doctype

Tüm XML eşleme kütükleri bir Doctype bildirmelidirler.

5.1.2 hibernate-mapping

```
<hibernate-mapping
schema="schemaName"
catalog="catalogName"
default-cascade="cascade_style"
default-access="field|property|ClassName"
default-lazy="true|false"
auto-import="true|false"
package="package.name"
/>
```

schema: veri tabanına ait şema adı.İsteğe bağlıdır.

default-cascade:basamaklama yöntemini belirtmeyen alanlar için hangi basamaklama yönteminin kullanılacağını belirtir. İsteğe bağlı olup önbelirli *none* olarak tanımlanmıştır.

auto-import :sorgu dilinde sınıf isimlerini niteleyicisiz(unqualified) olarak kullanılmasını belirler(bu eşlemedeki sınıflar için geçerlidir) . İsteğe bağlı olup önbelirli *true* olarak tanımlanmıştır.

package: niteleyicisiz sınıf isimleri için paket adını bir önek olarak kullanmak üzere belirlenir.İsteğe bağlıdır.

Belgeye ilişkin eleştiri ve önerilerinizi lütfen Ar. Gör. Fatih M. GÜLEÇ'e bildirin.
fatihg@cs.hacettepe.edu.tr

5.1.3 class

```
<class
name="ClassName"
table="tableName"
discriminator-value="discriminator_value"
mutable="true|false"
schema="owner"
catalog="catalog"
proxy="ProxyInterface"
dynamic-update="true|false"
dynamic-insert="true|false"
select-before-update="true|false"
polymorphism="implicit|explicit"
where="arbitrary sql where condition"
persister="PersisterClass"
batch-size="N"
optimistic-lock="none|version|dirty|all"
lazy="true|false"
entity-name="EntityName"
check="arbitrary sql check condition"
rowid="rowid"
subselect="SQL expression"
abstract="true|false"
node="element-name"
/>
```

name: Kalıcı sınıfın tam java adıdır.

table: Veri tabanı tablo adıdır.

discriminator-value : Alt sınıfları ayırt etmek için kullanılan değer. İsteğe bağlı olup önbelirli olarak sınıf adıdır.

mutable :Sınıfa ait nesnelerin mutable olup olmadığını belirtir. İsteğe bağlı olup önbelirli *true* olarak tanımlanmıştır.

schema: <hibernate-mapping> kesiminde verilen şema adını geçersiz kılar.

proxy : Öndeğer atama*(lazy initializing)* için bir ara yüz belirler.

dynamic-update UPDATE SQL komutunun dinamik olarak çalışma zamanında oluşturulmasını denetler. İsteğe bağlı olup önbelirli *false* olarak tanımlanmıştır.

dynamic-insert : INSERT SQL komutunun dinamik olarak çalışma zamanında oluşturulmasını denetler. İsteğe bağlı olup önbelirli *false* olarak tanımlanmıştır.

Belgeye ilişkin eleştiri ve önerilerinizi lütfen Ar. Gör. Fatih M. GÜLEÇ'e bildirin.
fatihg@cs.hacettepe.edu.tr

select-before-update: UPDATE komutu çalıştırılmadan önce SELECT çalıştırılarak gerçekten nesne üzerinde değişiklik var mı yok mu belirlenmesini sağlar. İsteğe bağlı olup önbelirli *false* olarak tanımlanmıştır.

polymorphism : *Polimorfik* sorguların açık veya örtülü olarak gerçekleştirilmesini sağlar. İsteğe bağlı olup önbelirli *implicit* olarak tanımlanmıştır.

where SQL WHERE koşulunun kullanılıp kullanılmamasını belirler.

persister : Bir *ClassPersister* seçmeyi sağlar.

batch-size :*Batch size* 'ı ayarlamamızı sağlar. İsteğe bağlı olup önbelirli *1* olarak tanımlanmıştır.

optimistic-lock :İyimser kilitleme yönteminin kullanılmasını sağlar. İsteğe bağlı olup önbelirli *version* olarak tanımlanmıştır.

lazy: "true" olarak belirlemek proxy ara yüzü olarak sınıfın kendi ismini kullanmakla aynı duruma tekabül eder.

5.1.4 id

```
<id
name="propertyName"
type="typename"
column="column_name"
unsaved-value="null|any|none|undefined|id_value"
access="field|property|ClassName">
node="element-name|@attribute-name|element/@attribute|."
<generator class="generatorClass"/>
</id>
```

Eşlenen sınıflar veri tabanı tablosu için bir birincil anahtar alanı tanımlamalıdırlar. <id> alanı sınıfın tanımladığı ayırt edici alanın veri tabanına eşlenmesi tarif eder.

name : Ayırt edici alanın adıdır

type : Hibernate veri türünü belirtir

column: Birincil anahtar kolonunun adıdır. İsteğe bağlı olup önbelirli *name* olarak tanımlanmıştır.

unsaved-value: Yeni oluşturulmuş ve kaydedilmemiş nesnenin daha önceden oluşturulmuş nesnelerden ayırt edilmesini belirler. İsteğe bağlı olup önbelirli *null* olarak tanımlanmıştır.

access: Hibernatin property'e erişmesi için stratejiyibelirler. İsteğe bağlı olup önbelirli property olarak tanımlanmıştır.

Belgeye ilişkin eleştiri ve önerilerinizi lütfen Ar. Gör. Fatih M. GÜLEÇ'e bildirin.
fatihg@cs.hacettepe.edu.tr

5.1.5 property

property kalıcı alanlar tanımlamayı sağlar.

```
<property
name="propertyName"
column="column_name"
type="typename"
update="true|false"
insert="true|false"
formula="arbitrary SQL expression"
access="field|property|ClassName"
lazy="true|false"
unique="true|false"
not-null="true|false"
optimistic-lock="true|false"
generated="never|insert|always"
node="element-name|@attribute-name|element/@attribute|."
index="index_name"
unique_key="unique_key_id"
length="L"
precision="P"
scale="S"
/>
```

name: küçük harflerle alan adı

column:veri tabanındaki karşılık kolon adı. İsteğe bağlı olup önbelirli *name* olarak tanımlanmıştır.

type : Hibernate veri türünün adı

update, insert : Eşlenecek kolonların select ve/veya update sorgularında yer alıp almamasını belirler. İsteğe bağlı olup önbelirli *true* olarak tanımlanmıştır.

formula: Veri tabanındaki mevcut kolonlar kullanılarak hesaplanabilecek alanı belirtir. Bu alan aslında herhangi bir tabloda mevcut değildir.

access : Hibernatin property'e erişirken kullanacağı stratejiyi belirler. İsteğe bağlı olup önbelirli *property* olarak tanımlanmıştır.

5.1.6 many-to-one

bir çoğa-bir ilişkiyi tanımlamak için kullanılır. Bu sınıfın birçok nesnesinin başka bir sınıfın bir nesnesiyle ilişkili olduğunu belirtir. Alanların ayrıntılı açıklamalarına yer verilmemiştir.

```
<many-to-one
name="propertyName"
column="column_name"
class="ClassName"
cascade="cascade_style"
fetch="join|select"
update="true|false"
insert="true|false"
property-ref="propertyNameFromAssociatedClass"
access="field|property|ClassName"
unique="true|false"
not-null="true|false"
optimistic-lock="true|false"
lazy="proxy|no-proxy|false"
not-found="ignore|exception"
entity-name="EntityName"
formula="arbitrary SQL expression"
node="element-name|@attribute-name|element/@attribute|."
embed-xml="true|false"
index="index_name"
unique_key="unique_key_id"
foreign-key="foreign_key_name"
/>
```

5.1.7 one-to-one

bire-bir ilişkiyi tanımlamak için kullanılır.Bu sınıfın bir nesnesi başka bir sınıfın bir nesnesiyle ilişkilidir. Alanların ayrıntılı açıklamalarına yer verilmemiştir.

```
<one-to-one
name="propertyName"
class="ClassName"
cascade="cascade_style"
constrained="true|false"
fetch="join|select"
property-ref="propertyNameFromAssociatedClass"
access="field|property|ClassName"
formula="any SQL expression"
lazy="proxy|no-proxy|false"
entity-name="EntityName"
node="element-name|@attribute-name|element/@attribute|."
embed-xml="true|false"
foreign-key="foreign_key_name"
/>
```

5.1.8 subclass

Alt sınıfları ve ata sınıfla olan ilşkilerini belirler. Alanların ayrıntılı açıklamalarına yer verilmemiştir.

```
<subclass
name="ClassName"
discriminator-value="discriminator_value"
proxy="ProxyInterface"
lazy="true|false"
dynamic-update="true|false"
dynamic-insert="true|false"
entity-name="EntityName"
node="element-name"
extends="SuperclassName">
<property .... />
</subclass>
```

6-Hibernate Sorgulama Dili (HQL)

Hibernate'te kullanılan dil olan HQL genel olarak SQL ile benzerlikler gösterir. Yapısal benzerliklere rağmen temel de çok büyük bir farklılıkları mevcuttur.Bu SQL'in veri tabanı nesneleriyle çalışırken,HQL'in java sınıflarıyla başka bir deyişle nesneler üzerinde çalışmasıdır.

6.1.Büyük Küçük Harf Duyarlılık:

HQL'de sorgular genel olarak küçük büyük harf duyarsızdır. Sadece sorgularda kullanılan java sınıf adları, özellik adları veya nesne adlarında küçük büyük harf duyarlılık bulunmaktadır.
Mesela Select, SELEct, SELect gibi bu sorgu sözcüğündeki tüm olasılıklar sonuç olarak aynı değere sahiptirler. Ama bunun aksine "Hasta" isimli bir java sınıf tanımımız olsun burada HQL'de bu sınıf tanımını kullanmak istersek büyük küçük harf duyarlılığa dikkat etmeliyiz. "Hasta"'nın her hangi bir harfini değiştirdiğimizde karşımıza çıkan sonuçta farklılıklarla karşılarşabiliriz.

6.2.From yan Cümleciği

"From" veritabanından hangi sınıfın kopyalarının getirileceğini belirten yan cümledir. Sonuç olarak veritabanından sadece bir sınıfın değerlerini isteyebileceğimiz gibi birden çok sınıfın kartezyen çarpımlarını da isteyebiliriz.

Örneğin:

From Hasta: Hasta sınıfın tüm üyelerini veri tabanından bize getirir.
From Hasta, Hastane: Hasta ve Hastane sınıflarının kartezyen çarpımlarını bize getirir.

6.3.İlişkiler ve Join

HQL'de dört tip join vardır. Bunların özellikleri SQL deki joinlerle aynıdır.

- inner join
- left outer join
- right outer join
- full join(pek kullanılmaz)

6.4.Select Cümlesi

Select cümleleriyle sorgulardan o sınıfların hangi özelliklerinin döndürüleceği seçilir. Bu seçilen özellik sayısı sadece bir olabileceği gibi birden fazla ve ayrı sınıfların özellikleride olabilirler.

Örneğin:

```
Select Hastane.Adi, Hasta.Adi
From Hastane
Left outer join Hasta
```

Bu sorgunun sonucunda mesela hastane adları ve o hastanede bulunana hastaların adları yan yana listelenir.

Yine bu sorgulardan sadece tek bir nesne dönebilirken bazen bir nesneler diziside dönebilir. Bunun dışında bu sorgulardan dönen nesne özellikleriyle beraber yeni bir nesnede oluşturalabilir.

Örneğin:

```
Select new HastaneHastasi(Hastane.Adi,Hasta.Adi)
From Hastane
Left outer join Hasta
```

6.5.Where,Group by, Order by cümlecikleri

Where cümleciği getirilen sınıflar için bir seçicilik sağlar. Veritabanında o sınıfla ilgili tüm nesneler yerine where cümleciği ile kısıtlanmış şartları sağlayan nesneler getirilir.

Order by , sorgudan dönen nesnelerin hangi niteliğe/niteliklere göre sıralanacağını belirtmemizi sağlar.

Group by ise sorgudan dönen nesneleri, herhangi bir niteliğe göre gruplanmasını sağlar.

6.6.AltSorgular

Hibernate alt sorguları destekleyen veritabanı sistemleri için alt sorgu özelliğini destekler.

Örneğin:

```
Select Hasta.Adi from Hasta
Where Hasta.Kilo > ( select avg(Hasta.Kilo) from Hasta )
```

7- Hibernate Uygulaması

7.1.Geliştirme Ortamının Hazırlanması

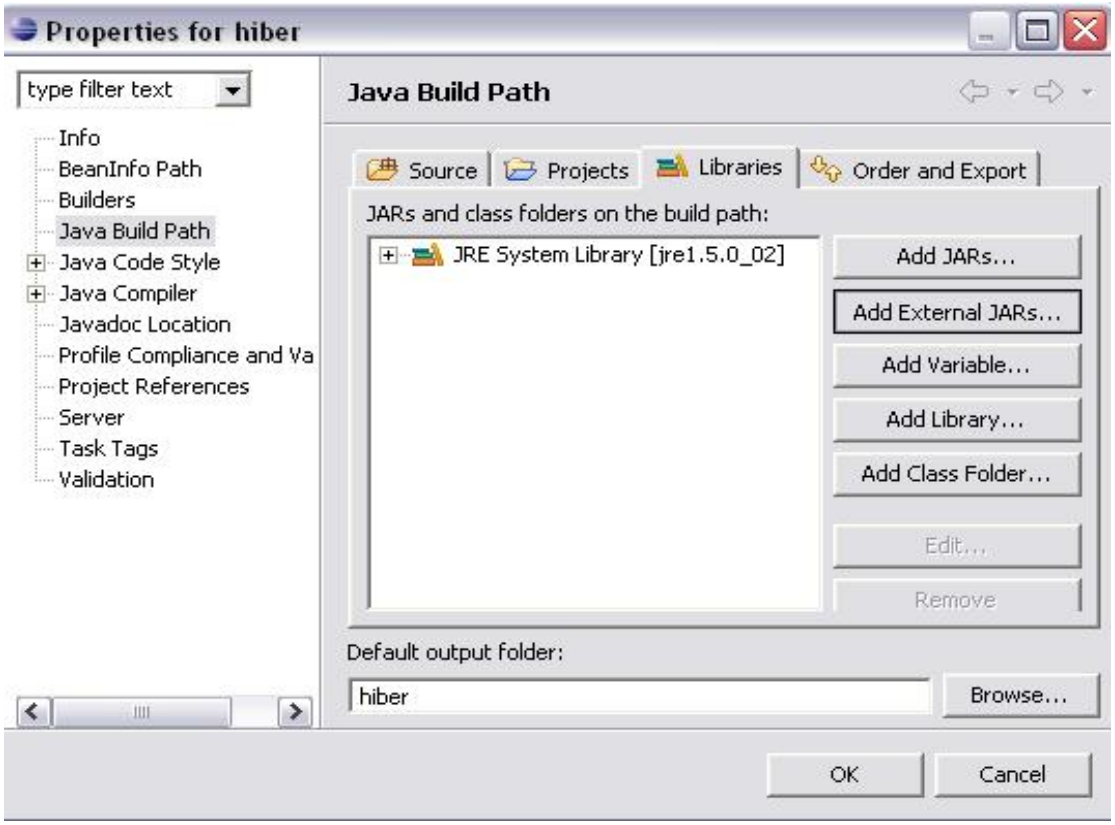
Örnek uygulamamızı gerçekleştirmeye uygun bir çalışma ortamı hazırlayarak başlayacağız.

Geliştireceğimiz uygulamada kullanılacak olan bileşenler şunlardır:

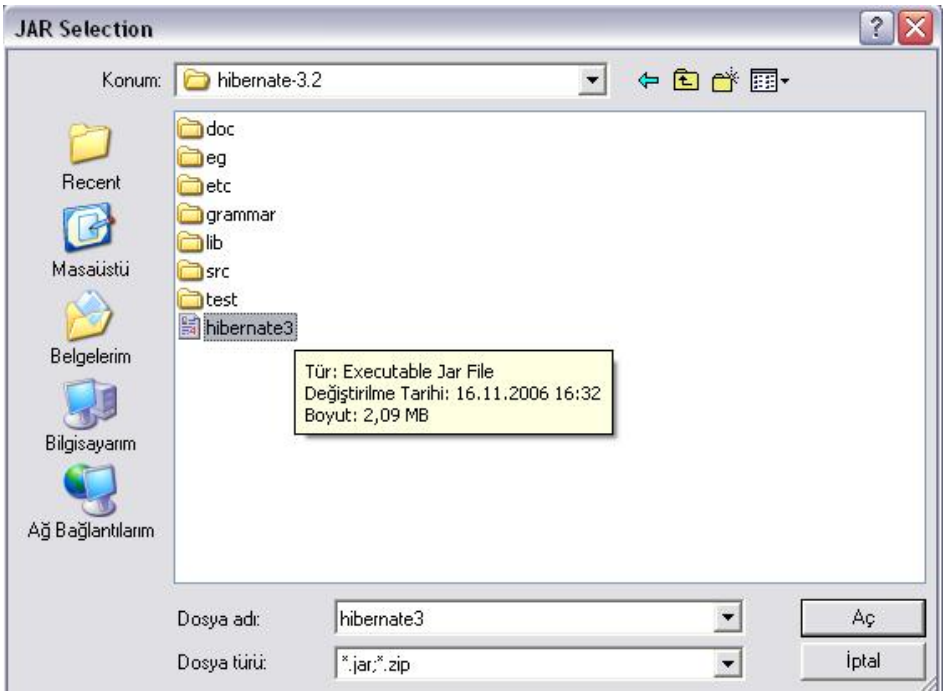
- JDK 1.5.0 sürümü kullanılmıştır. www.java.com adresinden indirilebilir.
- IDE olarak *Eclipse 3.2.1* kullanılmıştır. *Eclipse*'in bu sürümü www.eclipse.org adresinden indirebilirsiniz.
- Uygulamada kullanacağımız veri tabanı sunucusu ise *PostgreSQL 8.2.1* dir. Bu sürümü http://www.postgresql.org/ adresinden indirebilirsiniz.

Sonraki aşamada eğer kullandığınız platform *hibernate* kütüphanelerini içermiyorsa *hibernate* kütüphanelerini elde etmek için http://www.hibernate.org/ adresinden *hibernate'i* temin edebilirsiniz.

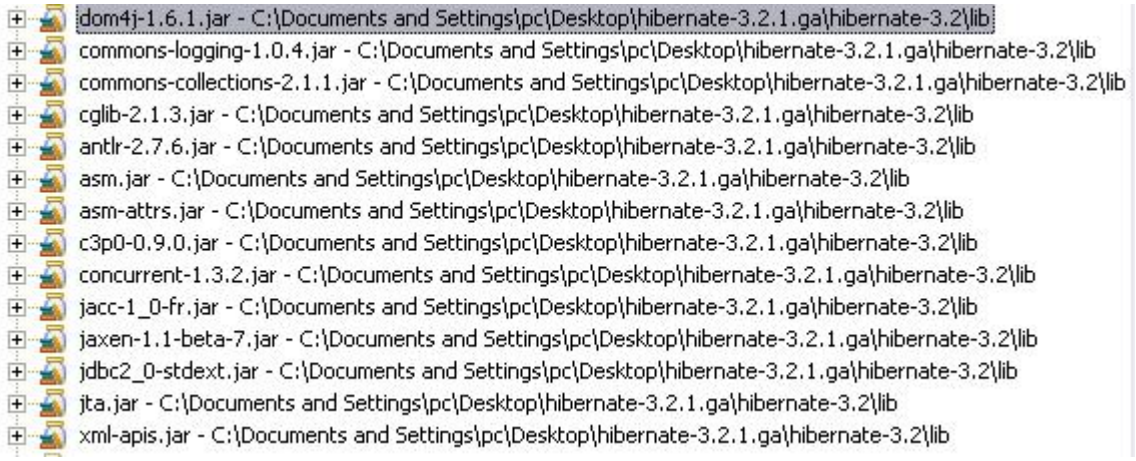
Hibernate herhangi bir yere çıkarttıktan(extract ettikten) sonra \hibernate-3.2\lib dizinindeki kütüphaneleri projenize eklemeniz gerekmektedir.Bunun için Project->Properties-> Java Build Path->Add External Jars yolunu izlemeniz gerekmektedir.



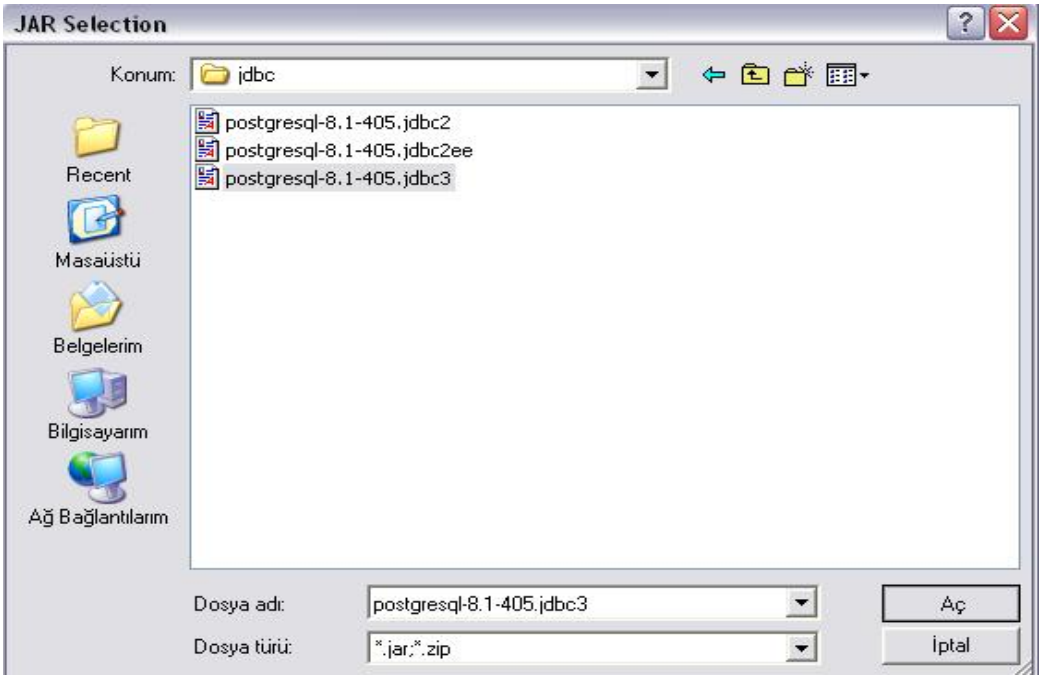
Burada eklememiz gereken kütüphanelerden biri hibernate'i çıkarttığımız dizinde bulunan hibernate3 kütüphanesidir.



Daha sonra aynı şekilde /lib dizinindeki ihtiyaç duyduğunuz kütüphaneleri ekleyebilirsiniz.Bizim örnek uygulamamızdaki aşağıdaki kütüphaneler kullanılmaktadır.



Eklememiz gereken son kütüphane ise kullandığımız veri tabanı sunucusuna bağlanmamızı sağlayacak kütüphanedir.Bu kütüphaneyi postgreSQL için postgreSQL'i kurduğunuz dizinde *\PostgreSQL\8.1\jdbc* dizini altında bulabilirsiniz.



Bu noktada projemizin hibernate uygulaması geliştirmeye hazırdır.Yapılandırma kesiminde örnek veri tabanı ve yemek kalıcı sınıfının oluşturulması ve bunların birbirleriyle yapılandırma kütükleri sayesinde eşleştirilmesi anlatılacaktır.

7.2.Hibernate Yapılandırma Kütüklerinin Hazırlanması

Yemek örneğimiz bir sınıf , bir adresleme kütüğü ve hibernate için tanımlanacak bir yapılandırma dosyası içermektedir.Yemek sınıfının genel yapısı aşağıda verilmiştir.

Yemek.java

```
package Objects;

public class Yemek {

	private int id; //belirleyici nitelik
	private String yemekadi;
	private int fiyati;

	public Yemek(){
	}
	public Yemek(int id, String yemekadi, int fiyati) {
		this.id = id;
		this.yemekadi = yemekadi;
		this.fiyati = fiyati;
	}
	public int getFiyati() {
		return fiyati;
	}
	public void setFiyati(int fiyati) {
		this.fiyati = fiyati;
	}
	public int getId() {
		return id;
	}
	public void setId(int id) {
		this.id = id;
	}
	public String getYemekadi() {
		return yemekadi;
	}
	public void setYemekadi(String yemekadi) {
		this.yemekadi = yemekadi;
	}
}
```

Java'da kalıcılığı sağlayan tüm çatılar(framework) gibi hibernate de verilerin Javabean tarzında yazılmış olmasını bekler.JavaBean tarzı sınıf demek tüm nitelikleri private olan, belirleyici niteliği sıralı(serialized) olan, niteliklerine ulaşmak için get/set methodları tanımlayan ve boş bir yapıcısı bulunan sınıf demektir. Dolayısıyla yemek sınıfı javabean tarzında verilmiştir.

Yemek sınıfımızın 3 niteliği vardır.Bunlar biri belirleyici nitelik olmak üzere yemek adı ve fiyatı alanlarıdır.Belirleyici nitelik,uygulamanın veri tabanını kimliğine yani veri tabanındaki kalıcı nesneye erişmesini sağlar.

Uygulamadan aynı zamanda yaratılan yemekleri veri tabanında kalıcı veri olarak da tutmasını bekliyoruz.Bunun için öncelikle veri tabanında Yemek tablosu oluşturmamız gerekmektedir.Aşağıda verilen SQL cümleciği Yemek veritabanını oluşturmamızı sağlayacaktır.

```
Create table "Yemek"
(
     "id" Integer NOT NULL UNIQUE,
     "fiyati" Integer NOT NULL,
     "yemekadi" Char(20) NOT NULL,
 primary key ("id")
)
```

PostreSQL Veri tabanında yemek tablosunu oluşturduktan sonra hibernate'e uygulamamızdaki yemek sınıfı ile veri tabanındaki yemek tablosunun eşleştiğini bildirmemiz gerekmektedir .Bu bilgi genellikle daha önce de belirtildiği üzere bir *xml eşleme kütüğü*nde (xml mapping document) sağlanır.Eşleme kütüğü her şeyden önce yemek sınıfının özelliklerini veri tabanında ki yemek tablosunun sütun adlarıyla eşler.
Bu örneğimiz için xml eşleme kütüğü şöyle olmalıdır:

Yemek.hbm.xml

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
"-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping package="Objects">
     <class name="Yemek" table="yemek">
          <id name="id" type="java.lang.Integer">
               <column name =”id”>
               <generator class=”increment”></generator>
          </id>
          <property name="yemekadi">
           <column name="yemekadi" />
          </property>
          <property name="fiyati">
          <column name="fiyati"/>
          </property>
     </class>
</hibernate-mapping>
```

Bu eşleme kütüğü Hibernate'e yemek sınıfının yemek tablosuyla kalıcılılaştırıldığını söylemektedir.*Id* niteliği sınıfımızın belirleyici niteliği olup tablonunda birincil anahtarıdır görüldüğü üzere *id* niteliği yeni bir nesne yaratırken girilmek zorunda değildir zira hibernate yemek nesnelerine artan şekilde bir *id* atayacaktır . Veri tabanında saklamak istediğimiz her sınıf için bu

Belgeye ilişkin eleştiri ve önerilerinizi lütfen Ar. Gör. Fatih M. GÜLEÇ'e bildirin.
fatihg@cs.hacettepe.edu.tr

adresleme kütüğü tanımlanmak zorundadır.Bu eşleme kütükleri kalıcı sınıflarla aynı dizinde bulunmak zorundadır

Geliştirdiğimiz bu küçük çaplı uygulamanın yapılandırma dosyasının tam hali aşağıda verilmiştir.

Hibernate.cfg.xml

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
<session-factory>
<property
name="hibernate.connection.driver_class">org.postgresql.Driver</prope
rty>
<property
name="hibernate.connection.url">jdbc:postgresql://localhost/deneme</p
roperty>
<property name="hibernate.connection.username">postgres</property>
<property name="hibernate.connection.password">123456</property>
<property name="hibernate.connection.pool_size">10</property>
<property name="show_sql">true</property>
<property
name="dialect">org.hibernate.dialect.PostgreSQLDialect</property>
<property
name="transaction.factory_class">org.hibernate.transaction.JDBCTransa
ctionFactory</property>

<!-- Mapping files -->
<mapping resource="Objects/yemek.hbm.xml"/>
</session-factory>
```

Yapılandırma dosyası,uygulamada kullanılacak eşleme(mapping) dosyalarını tanımlar.Bu tanımlama işlemi şöyle olmaktadır.Uygulamamızda kalıcılığıyla ilgilendiğimiz tüm sınıfların eşleme kütükleri aşağıda ki formatta yapılandırma dosyasına yazılır.

```
<mapping resource="Objects/yemek.hbm.xml"/>
```

Yapılandırma dosyasının yerine getirdiği diğer bir işlev, veri tabanı ile ilgili bilgileri de içermesidir.Dolayısıyla,kullanılan veri tabanı programı değiştiği zaman yapılması gereken, bu kütükteki ilgili birkaç satırın yeni veri tabanı programının nitelikleriyle değiştirilmesidir. Bu kolaylık hibernate'in en önemli avantajlarından biridir. Yapılandırma dosyasının veri tabanı ile ilgili kesimi şöyledir.

```
<session-factory>
<property
name="hibernate.connection.driver_class">org.postgresql.Driver</prop
erty>
<property
name="hibernate.connection.url">jdbc:postgresql://localhost/deneme</
property>
<property name="hibernate.connection.username">postgres</property>
<property name="hibernate.connection.password">123456</property>
<property name="hibernate.connection.pool_size">10</property>
<property name="show_sql">true</property>
<property
name="dialect">org.hibernate.dialect.PostgreSQLDialect</property>
<property
name="transaction.factory_class">org.hibernate.transaction.JDBCTrans
actionFactory</property>
```

Görüldüğü üzere veri tabanında oturum açma işlemi ile ilgili düzenlemeler yapılandırma dosyasında yapılmaktadır.yeni bir oturum açılacağı zaman bu dosya da yapılan bildirimlere göre tanımlı veri tabanı programına, tanımlı kullanıcı adı ve şifreyle bağlanılır.Veri tabanı programı değiştirilmek istenince yapılması gereken tek şey *dialect* alanına yeni programın dialectini yerleştirmek ve ilgili kullanıcı adı,şifre alanlarını değiştirmektir ve ilgili veri tabanının kütüphanelerini projeye dahil etmektir.Ayrıca bağlantı havuzunun boyutu da görüldüğü gibi yapılandırma dosyasında tanımlanmıştır.

```
<property
name="transaction.factory_class">org.hibernate.transaction.JDBCTrans
actionFactory</property>
```

Yukarıda ki kesimde ise hareketlerin nasıl ele alınacağı uygulamaya bildirilir.Örneğin burada hareketlerin altta yatan JDBC katmanına aktarılacağı bildirilmiştir.

Bu noktada projemizin hibernate uygulaması geliştirmeye hazırdır.Örnek uygulama kesiminde hibernate'in kullanımına yönelik bilgiler verilecektir.Hibernate ile tanımladığımız *yemek* tablosuna yemek ekleme,silme,güncelleme örnekleri verilecektir.

Belgeye ilişkin eleştiri ve önerilerinizi lütfen Ar. Gör. Fatih M. GÜLEÇ'e bildirin.
fatihg@cs.hacettepe.edu.tr

7.3.Örnek Uygulama

Bu kesimde örnek bir test sınıfı oluşturulup veri tabanı üzerinde ekleme, silme, güncelleme gibi basit veri tabanı işlemleri yapılacaktır.

Öncelikle uygulamada veri tabanına bağlanmak için yazılması gereken genel çatı aşağıdaki gibidir. Veri tabanı üzerinde yaptığınız ekleme, silme,veri çekme gibi işlemler bu kod bloğunun içine yazılmalıdır.Bu kod bloğu bize veri tabanına erişmek için bir bağlantı sağlamaktadır.

```
Session session = null;
SessionFactory sessionFactory = new Configuration().
configure().buildSessionFactory();
session =sessionFactory.openSession();
org.hibernate.Transaction trans =session.beginTransaction();

//hibernate işlemleri

trans.commit();
session.close();
```

Bu kod kesimi Hibernate'in *session* ve *transaction* arayüzlerini çağırır.Bunun için ilk olarak Sessionfactory'den yeni bir oturum talep edilmiş daha sonra yemek yaratma ve eklenme işlemleri bir *transaction* olarak gerçekleştirilip oturumu sonlandırmıştır

Veri tabanına yeni bir kalıcı yemek nesnesi eklemek için ise aşağıda ki kod kesimi yeterlidir.

```
Yemek yemek = new Yemek();
yemek.setYemekadi("Makarna");
yemek.setFiyati(10);
session.save(yemek);
```

.Bu kod bloğunun yaptığı iş aşağıda verilen SQL cümleciğininki ile aynıdır.

```
insert into Yemek(yemekadi,fiyati) values ('Makarna', 10);
```

Örneğimizde ki belirleyici nitelik olan *id* niteliği biricik bir değer tutmak zorundadır.Bu değer *save* komutu cağırıldığı zaman hibernate tarafından atanır. Daha önce de belirtildiği gibi bunun sağlanabilmesi için eşleme kütüğünde *id* niteliği üretecinin(generator) ,artan(increment) olarak atanması gerekir.

Böylelikle kalıcı yemek nesnemiz yaratılıp veri tabanına eklenmiş oldu.Bundan sonra ki amacımız kalıcı verimizi veri tabanından uygulamaya çekip kullanmak ise aşağıda ki kod işimizi görecektir.

```
Criteria criteria =
session.createCriteria(Yemek.class).addOrder(Order.asc("id"));
List yemekler = criteria.list();
for ( Iterator iter = yemekler.iterator(); iter.hasNext(); )
{
        Yemek y = (Yemek) iter.next();
        System.out.println( y.getYemekadi() );
}
```

Bu kod kesiminde veri tabanında ki bütün Yemek nesneleri uygulamaya çekilmiş ve yemek adına göre sıralayarak ekrana yazmıştır.Bu kod kesiminin yaptığı iş aşağıda verilen sql sorgusunun yaptığı işle aynıdır.

```
Select yemekadi from Yemek order by yemekadi asc;
```

Veri tabanında ki herhangi bir yemeği güncellemek için yazılması gereken kod kesimi ise aşağıdadır.Görüldüğü üzere JDBC de yaptığımız gibi bir SELECT sorgusuyla veri tabanından ilgili satırı çekip nitelikleri uygun nesneye teker teker atmaktansa , hibernate sayesinde veri tabanınından ilgili satırı direk nesne olarak elde edebiliriz.

```
yemek = new Yemek();
session.load(yemek,21);
yemek.setYemekadi("imambayildi");
session. session.saveOrUpdate(yemek);
```

Bir yemek nesnesini veri tabanından silmek için ise aşağıda ki kod kesimi yeterlidir.

```
yemek = new Yemek();
yemek.setId(21);
sess.delete(yemek);
```

Görüldüğü üzere java kodlarında herhangi bir sql cümleciği yazılmamıştır.SQL sorguları uygulama çalışırken arka planda hibernate tarafından oluşturulur.

Örnek test sınıfımızın tam hali aşağıdaki gibidir.Bu uygulama veri tabanına yeni bir yemek ekleyip veri tabanındaki yemekleri listelemektedir.

Belgeye ilişkin eleştiri ve önerilerinizi lütfen Ar. Gör. Fatih M. GÜLEÇ'e bildirin.
fatihg@cs.hacettepe.edu.tr

Main.java

```
package Objects;
import java.util.Iterator;
import java.util.List;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
import org.hibernate.Criteria;
import org.hibernate.criterion.Order;

public class Main {
     public static void main(String[] args) {
          Session session = null;
          SessionFactory sessionFactory = new Configuration().
                              configure().buildSessionFactory();
          session =sessionFactory.openSession();
          org.hibernate.Transaction trans =session.beginTransaction();

          Yemek yemek = new Yemek();
          yemek.setYemekadi("Pizza");
          yemek.setFiyati(10);
          session.save(yemek);

          System.out.println(yemek.getYemekadi()+" veri tabanına
                               eklendi");

          Criteria criteria = session.createCriteria(Yemek.class)
                              .addOrder(Order.asc("id"));
          List yemekler = criteria.list();

         for ( Iterator iter = yemekler.iterator(); iter.hasNext(); ){
               Yemek y = (Yemek) iter.next();
               System.out.println( y.getYemekadi() );
          }
          trans.commit();
          session.close();

     }

}
```

Programı eclipse üzerinde çalıştırdığınız zaman aşağıdaki ekranda aşağıdaki çıktı üretilecektir.

```
Hibernate: select max(id) from yemek
Pizza veri tabanına eklendi
Hibernate: insert into yemek (yemekadi, fiyati, id) values (?, ?, ?)
Hibernate: select this_.id as id0_0_, this_.yemekadi as Yemekadi0_0_
,this_.fiyati as fiyati0_0_ from yemek this_ order by this_.id asc
Pizza
```

8-Kaynaklar

- *Hibernate in Action/* Christian Bauer & Gavin King
- *Hibernate Reference Documentation*
- *http://www.roseindia.net/hibernate/firstexample.shtml*
- *http://ftp.cs.hacettepe.edu.tr/pub/dersler/BIL4XX/BIL447_YML/belgeler/Hibernate.pdf*
- *https://www.ribesoftware.com/mpo/samples/java.hibernate.flex/htmlsrc.hibernate.cfg.xml.html*
- *http://ab.org.tr/ab06/bildiri/46.pdf*
- *http://www.hibernate.org/hib_docs/v3/reference/en/html/tutorial.html*
- http://www.onjava.com/pub/a/onjava/2004/01/14/hibernate.html