Domain Model ve Temel Alanların Eşleştirilmesi

Persistent Sınıfların Yazılması

- Persistent domain sınıflarının Hibernate'e özel herhangi bir **arayüz implement etmelerine gerek yok**tur
- Herhangi bir **sınıftan da türemez**ler
- **Persistence katmanı** dışında da kullanılabilirler (birim testleri, kullanıcı arayüzleri vs)
- **Serializable** arayüzünü implement etmek de zorunlu değildir

Persistent Sınıfların Yazılması

- En azından “**package visibility**”ye sahip **default constructor**'a ihtiyaç vardır
- Proxy üretimi nedeni ile **sınıfların final olmaması veya final metodların yer almaması** gerekir
- Anotasyonlar field düzeyinde tanımlanabilir
- Dolayısı ile **accessor metodları**nı non-public yapmak veya tamamen silmek mümkündür
- Bu metodlara **iş mantığına özel** kod, validasyon vs. de eklenebilir

Persistent Nesne Türleri

- Hibernate/JPA persistent domain nesnelerini iki farklı kategoride ele alır
  - Entity nesneler
  - Bileşenler (Embeddable)

Entity Nesneler

- Sınıfın başına @**Entity** annotasyonu eklenerek tanımlanır
- Entity nesne kendine ait bir **veritabanı kimliğine sahip**tir (DB primary key)
- Bir veya daha fazla attribute @**Id** annotasyonu ile identifier olarak belirlenir
- Her entity kendi yaşam döngüsüne sahiptir; yaratılır, persist edilir, detached olabilir, silinebilir
- Diğer entity'lerden **bağımsız** var olabilir

Entity Nesneler

- Her **entity'nin bir ismi var**dır
- Default olarak bu **sınıf ismi**dir
- Uygulama içerisinde **aynı isimde iki tane entity olamaz**
- **@Entity** anotasyonunun **name** attribute'u ile entity'ye farklı bir isim verilebilir
- Entity ismi **sorgularda vs sıklıkla kullanılmakta**dır

```
@Entity(name="MyPet")
public class Pet {
    @Id
    private Long id;
    ...
}
```

Bileşenler (Value Nesneler)

- Bileşenlerin kendilerine ait **veritabanı kimliği yok**tur
- Bileşen tipleri identifier içermezler
- Bileşenler çalışma zamanında **mutlaka bir entity nesneye ait olmalı**dırlar
- **Yaşam döngüleri** ait oldukları entity ile aynıdır

```
@Embeddable
public class Address {
...
}
```

Temel Alanların Eşleştirilmesi

- Annotasyonlar kullanılırken **default olarak bütün alanlar** persistent'dir
- **@Basic** anotasyonu ile persistent alanlar explicit biçimde belirtilebilir, kullanımı opsiyoneldir
- **@javax.persistence.Transient** annotasyonu veya **transient** keyword'ü property'yi persistent olmaktan çıkarır
- XML eşlemede ise persistent alanlar **explicit biçimde belirtilmeli**dir

Temel Alanların Eşleştirilmesi

- Eğer property bir **Java tipi** ise (int,String, boolean, enum) **otomatik olarak persistent** kabul edilir
- Eğer property'nin sınıfı **@Embeddable ise bileşen (component)** olarak eşleştirilir
- Property tipi **Serializable ise değeri veritabanında binary stream biçimde** tutulur
- Bunların dışında bir tip olduğunda ve ORM eşlemesi yapılmamış ise **hata** ortaya çıkar

Temel Alanların Eşleştirilmesi

```
@Basic
public String getName() {
    return name;
}
```

Temel alanların tanımlanmasında kullanılan annotasyondur, kullanılmasada field persistent kabul edilir

```
<property name="name" type="string" not-null="true"
access="property">
     <column name="NAME" not-null="true"/>
</property>
```

XML'de persistent alanlar belirtilemelidir

Temel Alanların Eşleştirilmesi ve Optional Attribute

- JPA spec'e göre provider bu attribute'u destekliyorsa **optional=false** olarak tanımlanmış bir alan NULL ise hiç SQL ifadesi üretmeden hata verilmelidir

```
@Basic(optional = false)
@Column(nullable = false)
public String getName() {
    return name;
}
```

@Column anotasyonundaki nullable ise sadece şema oluşturmada kullanılmaktadır. JPA provider'ın runtime'daki davranışını etkilemez

MappedSuperClass

- Sınıflar bir takım **üst sınıflar**dan inherit edebilirler
- Bu üst sınıfları **persistent entity sınıf olarak tanımlamadan** sahip olduğu property'lerin eşlenmesi istenebilir
- Bu tür üst sınıflar **@MappedSuperClass** annotasyonu ile işaretlenmelidir
- Üst sınıflarda **map edilmek istenen property'ler**de Hibernate eşlemeleri yapılır
- Alt sınıflar, eşleme sırasında bu property'leri **persistent field olarak inherit** edecektir

MappedSuperClass

```java
@MappedSuperclass
public abstract class BaseEntity implements Serializable {
     @Id
     private Long id;
...
}
```

```java
@MappedSuperclass
public abstract class Person extends BaseEntity {
     @Column(name="FIRSTNAME")
     private String firstName;

     @Column(name="LASTNAME")
     private String lastName;
...
}
```

```java
@Entity
@Table(name="T_OWNER")
@AttributeOverrides({
@AttributeOverride(name="id",column=@Column(name="OWNER_ID")),
@AttributeOverride(name="firstName",column=@Column(name="FIRST_NAME")),
@AttributeOverride(name="lastName",column=@Column(name="LAST_NAME")),})
public class Owner extends Person {
     @Column(name="EMAIL")
     private String email;
...
}
```

İletişim

- Harezmi Bilişim Çözümleri
- Kurumsal Java Eğitimleri
- http://www.java-egitimleri.com
- info@java-egitimleri.com