آموزش فریمورک

Hibernate

مولف:مهندس افشین رفوآ
www.tahlildadeh.com

HIBERNATE

بسم الله الرحمن الرحیم

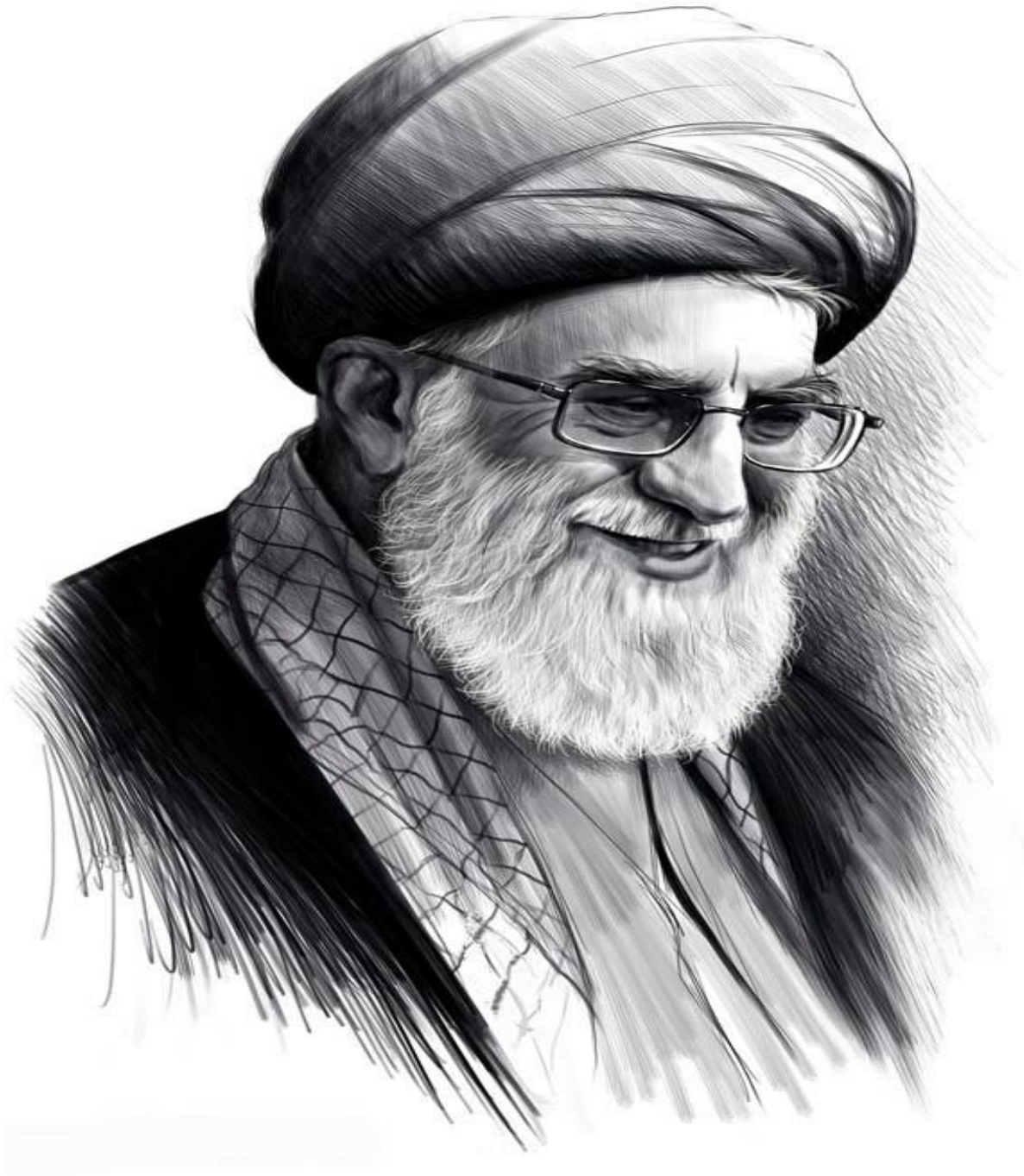
آموزشگاه تحلیل داده

تخصصی ترین مرکز برنامه نویسی و دیتا بیس در ایران

کتاب آموزش Hibernate جاوا

نویسنده : مهندس افشین رفوآ

آدرس آموزشگاه : تهران - خیابان شریعتی - بالا تر از خیابان ملک - جنب بانک صادرات - پلاک 651 طبقه دوم - واحد7
88146323 - 88446780 - 88146330



تقدیم به نائب امام عصر، حضرت آیت الله خامنه ای که عصا زدنش ضرب آهنگ حیدری دارد

آدرس آموزشگاه : تهران - خیابان شریعتی - بالا تر از خیابان ملک - جنب بانک صادرات - پلاک 651 طبقه دوم - واحد7
88146323 - 88446780 - 88146330

تقدیم به همه ی جویندگان علم که توان و امکان شرکت

در کلاس های ما را ندارند

TAHLILGAR DADEHA INSTITUTE
آموزشگاه تحلیلگر داده ها

TAHLILGAR DADEHA INSTITUTE
آموزشگاه تحلیلگر داده‌ها

فهرست

آشنایی با بخش آموزشHibernate 7
مزایای استفاده از چهارچوب کاریHibernate 7
آموزش معماریHibernate 8
آموزش المنت های اصلی معماری: Hibernate 10
آموزش ایجاد اولین برنامه Hibernaite بدون استفاده از محیطIDE 12
مرحله اول : ایجاد کلاس: Persistent Class 12
مرحله دوم : ایجاد فایل نقشه دهی mapping file برای کلاس: Presistent class 13
مرحله سوم : ایجاد فایل پیکربندی تنظیمات..... 14
مرحله چهارم : ایجاد کلاسی که object برنامه را ایجاد و نگهداری می کند: 15
مرحله پنجم : خواندن فایل: jar file 16
مرحله آخر : آموزش اجرای اولین برنامه Hibernate بدون استفاده ازIDE 16
آموزش ایجاد برنامه Hibernate در محیط: Eclips IDE 17
مرحله اول - ایجاد پروژه جاوا: 18
مرحله دوم - اضافه کردن فایل های jar به Hibernate..... 18
مرحله سوم - ایجاد کلاسpersistent 18
مرحله چهارم - ایجاد فایل mapping file جهت کلاسpersistent 19
مرحله پنجم - ایجاد فایل تنظیمات configuration file 20
مرحله ششم -ایجاد کلاس لازم جهت دریافت و نگهداری شیpersistent object 20
مرحله هفتم: اجرای برنامه Hiberate درEclipse 21
آموزش ایجاد برنامه Hibernate درMyEclipse 21
مرحله اول - ایجاد پروژه جاوا: 22
مرحله دوم - اضلفه کردن قابلیت های Hibernate به پروژه: 22
مرحله سوم - ایجاد کلاسPersistent 23
مرحله چهارم - ایجاد فایل نقشه دهی mapping file جهت کلاسPersistent 24
مرحله پنجم - ایجاد یک مسیر جهت فایل mapping در فایل تنظیماتConfiguration 25
مرحله ششم - ایجاد کلاس لازم جهت دریافت و نگهداری شی: persistent object 26
مرحله هفتم - اضافه کردن فایل jar لازم جهتoracle 26
مرحله آخر - اجرای برنامهHibernate 27
درس ششم : آموزش استفاده از annonation در Hibernate..... 28
مثال عملی ایجاد یک برنامه Hibernate به همراهAnnotation 28
مرحله اول-اضافه کردن فایل های jar لازم برای oracle وannotation 29

مرحله دوم-آموزش ایجاد کلاسPersistent Class 29
مرحله سوم-اضافه کردن mapping جهت کلاس Persistent در فایلConfiguration 30
مرحله چهارم-ایجاد کلاس لازم جهت تولید و نگهداری شیpersistent object 31
طراحی برنامه تحت وب باHibernate 31
مثال عملی ایجاد یک برنامه تحت وب باHibernate 32
آموزش کار با کلاس های سازندهHibernate Generator Class 34
درس نهم آموزش SQL Dialect درHibernate 38
درس دهم آموزش Hibernate Logging با Log4j و فایل xml در Hibernate 39
مراحل انجام عملیات loging به وسیله log4j با فایل xml : 40
مثال عملی انجام عملیات logging با استفاده از log4j و فایل xml : 40
آموزش Hibernate در Logging با Log4j و فایل properties 41
مراحل انجام عملیات logging با log4j و فایل propertiesدرHibernate 41
مثال عملی انجام عمل logging با استفاده از log4j و فایل ایل propertiesدرHibernate 41
آموزش آدرس دهی (mapping) درHibernate 42
جدول برمبنای سلسله مراتب(Table Per Hierarchy) 43
جدول برمبنای کلاس واقعی(Table Per Concerete Class) 43
جدول برمبنای کلاس های فرعی(Table Per Subclass) 43
آموزش مدل نگاشت Table Per Hierachy به وسیله فایلxml 44
مثال عملی کار با مدل نگاشتTable Per Hierarchy 45
آموزش مدل نگاشت Table Per Hierarchy به وسیلهAnnotation 49
مثال عملی نگاشت Table Per Hierarchy به وسیلهAnnotation 50
آموزش مدل نگاشت Table Per Concrete Class با استفاده از فایلxml 54
مثال عملی مدل نگاشت: Table Per Concrete Class 57
آموزش مدل نگاشت Table Per Concrete Class با استفاده ازAnnotation 60
مثال عملی مدل نگاشت: Table Per Concrete Class 62
آموزش مدل نگاشت Table Per Subclass با استفاده از فایلxml 65
مثال عملی نگاشت با مدلTable Per SubClass Class 68
آموزش مدل نگاشت Table Per Subclass با استفاده ازAnnotation 71
مثال عملی مدل نگاشت Table Per Subclass با استفاده ازAnnotation 73
آموزش آدرس دهی مجموعه ها (Collection) درHibernate 76
آموزش روش آدرس دهی مجموعه در فایلmapping 77
فهمیدن عنصرkey 79
مجموعه های ایندکس شده یاindexed collections 79

المنت های مجموعه ای یاCollection Elements 79
آموزش آدرس دهی المنت List در آدرس دهی مجموعه ها: 80
مثال عملی آموزش آدرس دهی المنت List در مجموعه ها 81
آموزش آدرس دهی به روش One to Many در Hibernate با استفاده از مثال List در فایلxml 84
مثال آدرس دهی (mapping) با روش one to many در Hibernate با استفاده ازList 86
آموزش نحوه دریافت (fetch) اطلاعات لیستList 89
آموزش Mapping Bag در آدرس دهی مجموعه هایHibernate 91
مثال عملی Mapping bag در آدرس دهی مجموعه هایHibernate 91
آموزش نحوه استخراج (fetch) اطلاعات: 94
آموزش آدرس دهی mapping One to Many در Hibernate با استفاده ازList 95
مثال عملی روش آدرس دهی one to many در مجموعه هایHibernate 96
آموزش نحوه دریافت fetch اطلاعات ازList 99
آموزش آدرس دهی متغیرهای Set در مجموعه هایHibernate 100
مثال عملی آموزش آدرس دهی متغیر Set در مجموعه هاHibernate 101
آموزش نحوه خواندن و دریافت fetch اطلاعات: 103
آموزش آدرس دهی متغیرهای Set با روش One to Many درHibernate 104
مثال عملی آدرس دهی Set در مجموعه های Hibernate با استفاده از روشone to many 105
آموزش آدرس دهی Map در مجموعه های Hibernate با استفاده از فایلxml 105
مثال عملی آدرس دهی Map در مجموعه های Hibernate با استفاده از فایلxml 106
آموزش آدرس دهی Many to Many Mapping در Hibernate با مثال map و استفاده از فایلxml 109
مثال عملی آموزش آدرس دهی Many to Many درHibernate 109
آموزش روابط دو طرفه Bidirectional Assocation درHibernate 114
آموزش کاربردHibernate Lazy Collection 115
آموزش آدرس دهی Component Mapping درHibernate 115
آموزش آدرس دهی Association Mapping با روش one-to-one درHibernate 117
آموزش آدرس دهی Association Mapping با روش one-to-one در – Hibernate بخش دوم 121
آموزش مدیریت تراکنش ها Tranaction Managment درHibernate 125
رابط کاربری تراکنش یا Transaction Interface درHibernate 126
مثال و کد عملی آموزش مدیریت تراکنش ها Tranaction Managment درHibernate 127
آموزش زبان کار با پایگاه داده HQL درHibernate 128
مزایای استفاده ازHQL 128
رابط کاربریQuery Interface 128
کد مثال دریافت اطلاعات کلید رکوردها با استفاده ازHQL 129

کد مثال دریافت اطلاعات کلیه رکوردها با استفاده از HQL و صفحه بندی نتایج: 129
کد مثالHQL Update Query 129
کد مثالHQL delete Query 129
استفاده از توابع ریاضی درHQL 130
کد مثال محاسبه حقوق کلیه کارمندان..... 130
کد مثال یافتن بالاترین میزان حقوق در کارمندان..... 130
کد مثال یافتن کمترین میزان حقوق در کارمندان..... 130
کد مثال محاسبه تعداد کارمندان درHQL 130
کد مثال یافتن میانگین حقوق کارمندان درHQL 130
آموزش زبان HCQL درHibernate 130
مزایای استفاده ازHQL 131
رابط کاربریCriteria Interface 131
آموزش کلاس های محدودکننده Restriction Class درHQL 132
مثال کد نوشتن با زبان HCQL درHibernate 133
کد مثال خواندن تمامی رکوردها با استفاده ازHCQL 133
کد مثال خواندن رکوردهای 10 ام تا 20 ام درHCQL 133
کد مثال خواندن رکوردهایی که میزان salary آن ها بیشتر از 10000 است به وسیلهHCQL 133
آموزش HCQL باProjection 133
آموزشHibernate Named Query 133
آموزش Hibernate Named Query با استفاده ازannotation 134
مثال عملی آموزش Hibernate Named Query با استفاده ازannotation 134
مثال عملی آموزش Named Query با استفاده ازmapping file 137
آموزش قابلیت Caching درHibernate 138
آموزشFirst Level Cache 138
آموزشSecond level Cache 138
آموزش Second Level Cache درHibernate 139
3مرحله اضافی که بایستی برای اجرای Second level cache به وسیله EH cache انجام دهید، عبارتند از: 140
مثال عملی آموزش Second Level Cache درHibernate 140
آموزش ادغام برنامه های Struts وHibernate 144
مثال آموزش ادغام برنامه های Struts 2 وHibernate 144
آموزش ادغام برنامه های Spring وHibernate 148
مزایای استفاده چهارچوب کاری Spring باHibernate 149
درک مشکل عدم استفاده از Spring باHibernate 149

آموزش متدهای پرکاربرد کلاسHibernateTemplate 150
مثال عملی آموزش ادغام برنامه های Spring وHibernate 151
آموزش فعال کردن قابلیت ایجاد خودکار جدول ها (table creation) ، نشان دادنquery های SQL و: 156

TAHLILGAR DADEHA INSTITUTE
آموزشگاه تحلیلگر داده ها

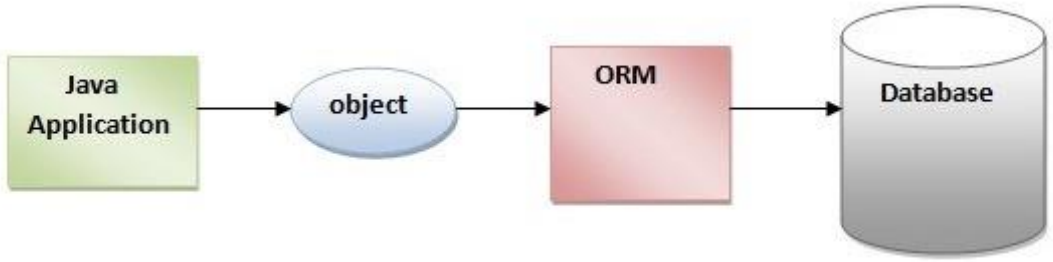
آشنایی با بخش آموزش Hibernate

در بخش آموزش Hibernate ، قصد داریم تا با ارائه مثال های ساده و عملی ، مفاهیم اصلی و کلیدی این تکنولوژی را به صورت کامل بررسی کنیم.

طراحی Hibernate در ارسال 2001 و توسط Gavin King به عنوان جایگزینی برای سیستم مدیریت Bean های جاوا یعنی EJB2 شروع شد . آخرین نگارش پایدار و نهایی شده Hibernate ، ورژن 4.3.6 است که در سال 2014 ارائه شده است . این تکنولوژی هم برای افراد مبتدی و هم برای مختصصان قابل استفاده و سودمند.

چهارچوب کاری Hibernate ، توسعه و نوشتن نرم افزارهای جاوایی که با پایگاه داده در تعامل هستند را ساده می کند Hibernate . یک ابزاراپن سورس ، کم حجم و سبک و دارای ویژگی ORM (Object Relational Mapping)است.

یک ابزار ORM، ایجاد ، دستکاری و دسترسی به داده ها را در سطح برنامه آسان می کند ORM . یک تکنیک برنامه نویسی است که اشیا (Object) های برنامه را به داده های موجود در پایگاه داده متصل و مرتبط میکند . شکل زیر ، رویه کار یک ORM را نشان می دهد:



ابزار ORM در درون خود از JDBC API برای ارتباط با پایگاه داده استفاده می کند.

مزایای استفاده از چهارچوب کاری Hibernate

استفاده از چهارچوب کاری Hibernate مزایای زیادی دارد که از آن جمله می توان به موارد زیر اشاره کرد:

1. اپن سورس و کم حجم بودن Hibernate : تحت لیسانس LGPL یک نرم افزار اپن سورس و در عین حال کم حجم و سبک است.

2. اجرای سریع : (fast performance) اجرای Hibernate به دلیل قابلیت Cacheدرونی آن ، بسیار سریع است . دو مدل عمل Cache در Hibernate وجود دادد : کش مرحله اول first level) (Cacheو کش مرحله دوم (second level Cache) که first level cache به صورت پیش فرض فعال است.

3. کوئری مستقل در دیتابیس: (Database Independent query)

HQLیا Language Hibernate Queryمدل شی گرای پایگاه داده SQL است . این زبان کوئری هایی مستقل از خود پایگاه داده تولید می کند . بنابراین نیاز ندارید queryرا مخصوص یک databaseبنویسید ، همچنین می توانید آنها را در پایگاه داده های مختلف نیز استفاده کنید . قبل از Hibernate ، اگر پایگاه داده تغییر می کرد ، مجبور بودید تمامی queryهای SQL را مجددا بازنویسی کرده تا در نگهداری و استفاده اطلاعات دچار مشکل نشوید .

4. تولید خودکار جدول های پایگاه داده Hibernate : امکان تولید جدول های پایگاه داده را به صورت اتوماتیک فراهم کرده است . بنابراین دیگر نیازی ندارید تا table های هر database را به صورت دستی بنویسید.

5. نوشتن ساده Join ها در : Hibernate دریافت و بازگرداندن اطلاعات از چندین جدول با استفاده از عمل Join در Hibernate بسیار ساده تر شده است.

6. فراهم نمودن قابلیت آمار query ها و وضعیت پایگاه داده Hibernate : از قابلیت Cache در Queryپشتیبانی می کند . همچنین این تکنولوژی قابلیت آمار گرفتن از کوئری های برنامه Query) (Statisticsو اطلاع از وضعیت پایگاه داده (database status) را فراهم نموده است.

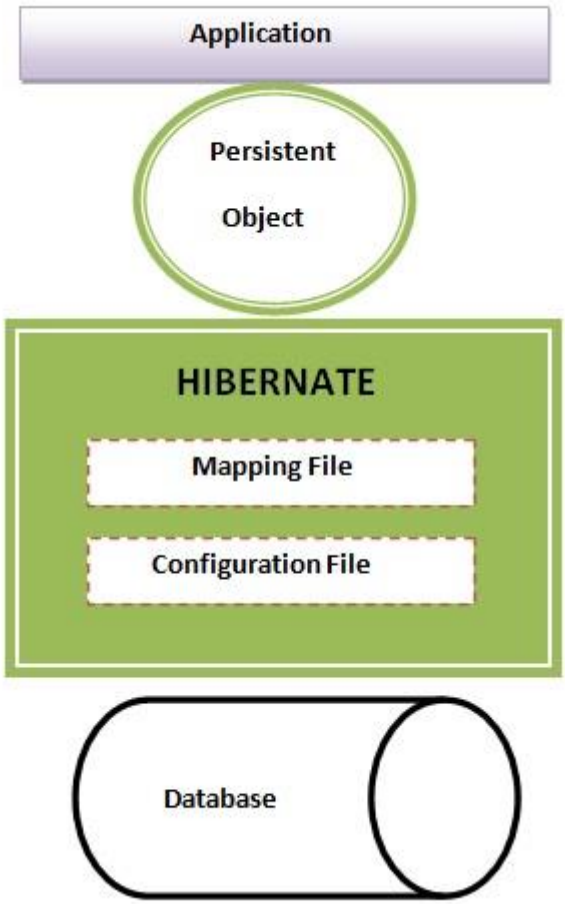
آموزش معماریHibernate

معماری Hibernate شامل اشیایی زیادی است که از آن جمله میتوان به presistent object، session factory، transaction factory، connection factory، session، transactionو ... اشاره کرد .

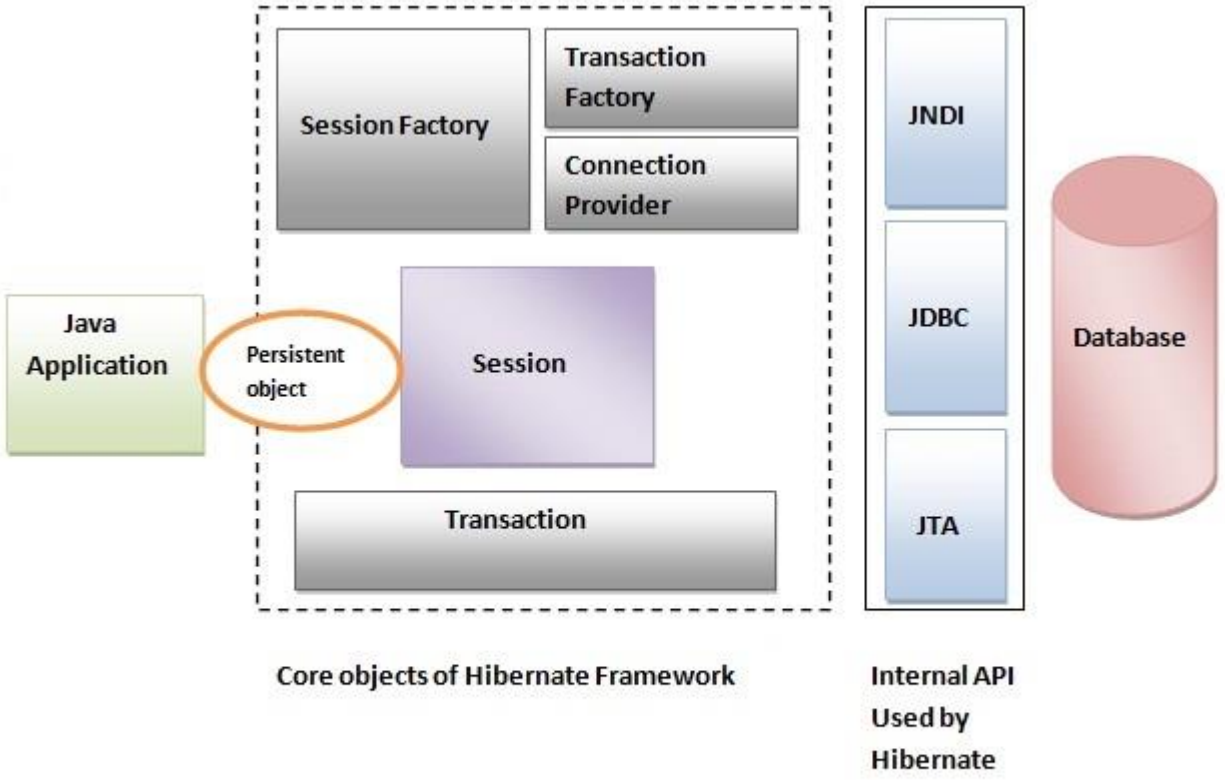
چهار لایه اصلی در معماری Hibernate وجود دارد که عبارتند از:

- java application layer
- hibernate framework layer
- backhand api layer
- database layer

شکل زیر لایه های مختلف معماری Hibernate و ارتباط آنها با یکدیگر را نشان می دهد:



شکل زیر بالاترین سطح معماری Hibernate به همراه mapping file و فایل تنظیمات configoration file را نشان می دهد:



همانطور که در مرحله قبل گفتیم ، Hibernateاز اشیای زیادی از جمله session fctory، transactionو ... در چهارچوب کاری خود استفاده می کند.

hibernateاز API های داخلی جاوا مثل Java database connectivity (JDBC) ، java transaction JTA((APIو (*) JNDI برای اتمام امور مختلف استفاده می کند.

آموزش المنت های اصلی معماری: Hibernate

برای ایجاد اولین برنامه به وسیله Hibernate ، بایستی اامنت های اصلی معماری ان را کاملا بشناسیم . این المنت ها عبارتند از:

- SessionFactory : المنت Session Factory در واقع یک کارخانه تولید sessionبرای برنامه بوده و مشتری سرویس connection proviver می باشد .یعنی اطلاعات مورد نیاز خود راجع به پایگاه داده را از connection proviver می گیرد

- SessionFactory* : المنت Session Factory در واقع یک کارخانه تولید session برای برنامه بوده و مشتری سرویس connection proviverمی باشد .یعنی اطلاعات مورد نیاز خود راجع به پایگاه داده را از connection proviver می گیرد .

- Session : المنت session object یک رابط کاربری بین نرم افزار و داده های موجود در databaseفراهم می کند . این شی یک object با عمر کوتاه (short-lived)بود و مقدار JDBC connectionرا مخفی می کند.

  این شی یک تولید کننده تراکنش ،کوئری و Criteria در برنامه است و از قابلیت cache سطح اول برای data استفاده می کند. رابط کاربری org.hibernate.session ، متد های لازم جهت insert ، updateو deletشی را فراهم می کند . این Interfaceهمچنین متد های لازم جهت تراکنش ها و کوئری ها نیز فراهم می کند .

- Transaction : یک شی transaction مشخص کننده مجموعه واحدی از عملیات ها در سیستم است .این مجموعه یا باید به طور کامل اجرا شود یا به طور کامل لغو گردد.استفاده از این شی اختیاری بوده و رابط کاربری org.transaction.hibernateمتد های لازم برای مدیریت تراکنش ها فراهم می کند.

- Connection Provider : این شی یک تولید کننده JDBC connection است . این شی برنامه را از data sourceو Drive managerجدا میکندو استفاده از ان اختیاری است.

- Transaction Factory : شی transaction factory هم تولید کننده تراکنش در سطح برنامه محسوب شده و استفاده از ان اختیاری است.

آموزش ایجاد اولین برنامه Hibernaite بدون استفاده از محیطIDE

در این درس ، قصد داریم تا نحوه طراحی یک برنامه Hibernate بدون استفاده از محیط IDE را آموزش دهیم.

برای ایجاد اولین برنامه Hibernate ، بایستی مراحل زیر را انجام دهید :

چهار لایه اصلی در معماری Hibernate وجود دارد که عبارتند از:

- ایجاد کلاس Persistent Class برنامه.
- ایجاد فایل نقشه دهی (mapping file) برای. Persistent Class
- ایجاد فتیل پیکربندی برنامه. Configuration file
- ایجاد کلاسی که شی Presistent object را ایجاد و نگهداری میکند .
- خواندن فایل. jar file
- اجرای برنامه Hibernate بدون استفاده از. IDE

مرحله اول : ایجاد کلاس: Persistent Class

یک کلاس ساده Persistent بایستی شامل اصل زیر باشد:

1. داشتن یک سازنده بدون آرگومان : (no-arg constructor) به شما توصیه می شود حتما یک تابع سازنده پیش فرض و بدون آرگومان برای کلاس داشته باشید . در این حالت Hibernate می تواند در هر زمان ، نسخه ای از کلاس Presistent class را با استفاده از متد ( ) New Instance بسازد .
2. تعیین یک خاصیت شناسایی کننده : (identifier property) این خاصیت به ستون کلید اصلی پایگاه داده متصل می شود .
3. تعیین متدهای setter و Hibernate : getter متدهای setter و getter کلاس را به صورت پیش فرض برحسب نام آنها ، شناسایی می کند .

4. ترجیح دادن کلاس های غیرنهایی برنامه Hibernate : (non-final class) از اصول proxy که برمبنای کلاس Presistent class تعیین شده اند ، استفاده می کند . بنابراین برنامه نویس Hibernateنمب تواند از proxy در هنگام انجام عملیات (Lazy feten دریافت کلاس ها به صورت خلاصه و ناکامل (استفاده کند .

کد :

در پایان ، کلاس Presistent class را با تعیین کد زیر جهت فایل Employee.javaایجاد کنید:

```
package com.javatpoint.mypackage;

public class Employee {
private int id;
private String firstName,lastName;

public int getId() {
    return id;
}
public void setId(int id) {
    this.id = id;
}
public String getFirstName() {
    return firstName;
}
public void setFirstName(String firstName) {
    this.firstName = firstName;
}
public String getLastName() {
    return lastName;
}
public void setLastName(String lastName) {
    this.lastName = lastName;
}


}
```

مرحله دوم : ایجاد فایل نقشه دهی mapping file برای کلاس Presistent class :

معمولا نام فایل نقشه دهی کلاس (mapping file) به صورت class_name.hbm.xmlاست . چندین المنت مختلف در این فایل قرار دارد که به بررسی آنها می پردازیم:

- المنت hibernate-mapping المنت root (مادر) در mapping file است.

- المنت class زیر را عنصر المنت hibernate-mapping می باشد . این المنت کلاس Presistent classرا مشخص می کند.
- ایجاد فتیل پیکربندی برنامه. Configuration file
- المنت generator زیر المنت المنت id است . این المنت جهت تولید کلید اصلی (primary key) به کار می رود .

  Generator classهای زیادی از جمله assigned ، increment، hilo، sequenceو ... وجود دارند که در ادامه و در بخش generator ها به بررسی آنها خواهیم پرداخت .
- المنت property هم یک زیر مجموعه از class است که خاصیت name مربوط به Persistent class را تعیین می کند.

کد :

در پایان مرحله دوم نیز فایل mapping file را با تعیین کد زیر جهت فایل employee.hbm.xml ایجاد نمایید:

```
<!--?xml version='1.0' encoding='UTF-8'?-->


 <hibernate-mapping>
  <class name="com.javatpoint.mypackage.Employee" table="emp1000">
    <id name="id">
     <generator class="assigned"></generator>
    </id>

    <property name="firstName"></property>
    <property name="lastName"></property>

  </class>

 </hibernate-mapping>
```

مرحله سوم : ایجاد فایل پیکربندی تنظیمات

Configuration fileشامل اطلاعات مختلفی راجع به پایگاه داده برنامه و mapping fileمی باشد . اصولا نام فایل تنظیمات به صورت hibernate.cfg.xml تعیین می شود:

کد:

```
<!--?xml version='1.0' encoding='UTF-8'?-->


<hibernate-configuration>

    <session-factory>
        <property name="hbm2ddl.auto">update</property>
        <property
name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
        <property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
        <property name="connection.username">system</property>
        <property name="connection.password">oracle</property>
        <property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
    <mapping resource="employee.hbm.xml">
    </mapping></session-factory>

</hibernate-configuration>
```

مرحله چهارم : ایجاد کلاسی که object برنامه را ایجاد و نگهداری می کند:

در این کلاس ، ما شی employee object را درون پایگاه داده ، تعریف و نگهداری می کنیم . به صورت زیر:

کد :

```
package com.javatpoint.mypackage;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;

public class StoreData {
public static void main(String[] args) {

    //creating configuration object
    Configuration cfg=new Configuration();
    cfg.configure("hibernate.cfg.xml");//populates the data of the
configuration file

    //creating seession factory object
    SessionFactory factory=cfg.buildSessionFactory();

    //creating session object
    Session session=factory.openSession();

    //creating transaction object
    Transaction t=session.beginTransaction();

    Employee e1=new Employee();
    e1.setId(115);
    e1.setFirstName("sonoo");
```

```
    e1.setLastName("jaiswal");

    session.persist(e1);//persisting the object

    t.commit();//transaction is commited
    session.close();

    System.out.println("successfully saved");

}
}
```

مرحله پنجم : خواندن فایل: jar file

برای اجرای موفقیت آمیز برنامه Hibernate ، شما بایستی فایل hibernate.jarرا در برنامه خودداشته باشید . از این آدرس می توانید آخرین نسخه فایل Hibernate jar را دانلود کنید برخی فایل های دیگر jar و packageهایی که مورد نیاز هستند عبارتند از:

- cglib
- log4j
- commons
- SLF4J
- dom4j
- xalan
- xerces

آدرس دانلود فایل های لازم

مرحله آخر : آموزش اجرای اولین برنامه Hibernate بدون استفاده از IDE

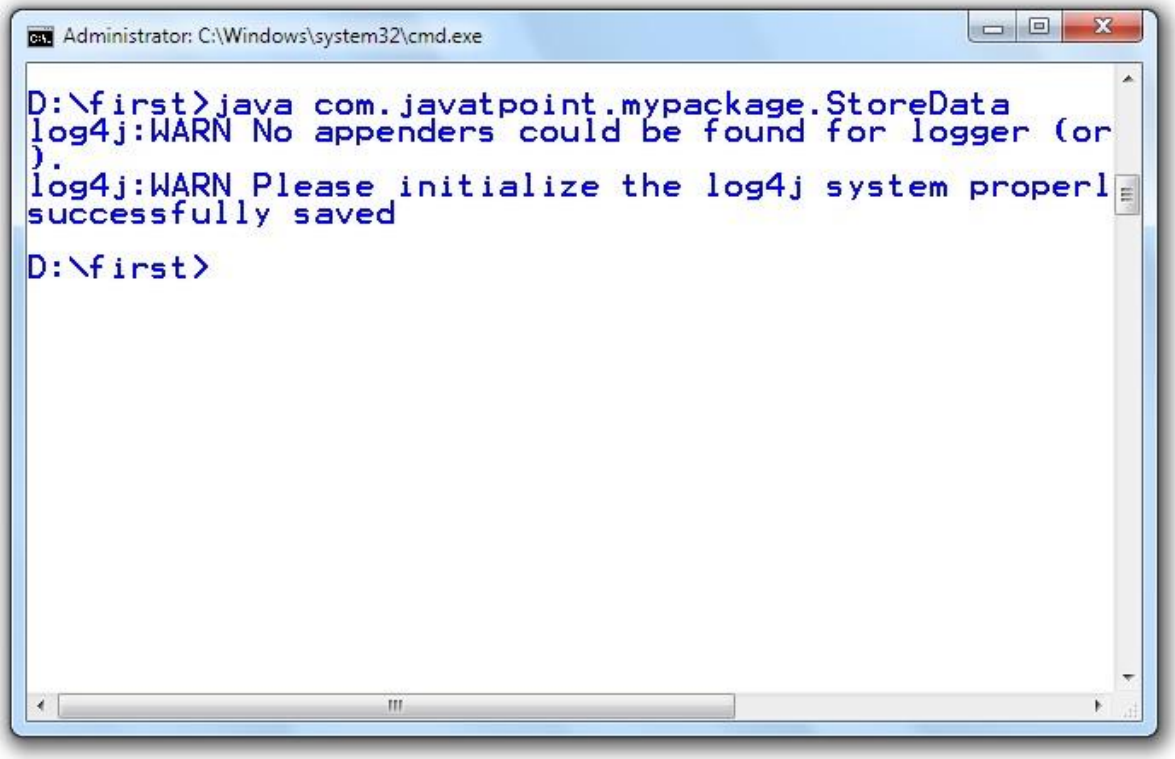
شما می توانید برنامه Hibernate خود را در یک IDE مثل Eclipse ، Net beansو ... اجرا کنید در مرحله بعد به آموزش نحوه اجرای یک برنامه Hibernateدر محیط Eclipse IDE خواهیم پرداخت.

اما برای اجرای یک برنامه Hibernate بدون استفاده از IDE ، بایستی مراحل زیر را انجام دهید:

1. Oracle logرا برای این مثال دانلود کنید.

2. فایل های jar برنامه را لود کنید . یکی از راه های لود فایل های jar ، کپی کردن آنها در مسیر پوشه JRE/lib/extاست . بهتر است فایل های jar را در هردو پوشه public و private پوشه JRE قرار دهید.

3. اکنون می توانید کلاس Store Data را با استفاده از java com.java point –my package.Store Dataاجرا کنید . همانند تصویر زیر:



آموزش ایجاد برنامه Hibernate در محیط: Eclips IDE

در این درس ، قصد داریم تا نحوه ایجاد و توسعه یک برنامهHibernate را در نرم افزار Elipe اموزش دهیم .

برای ایجاد اولین برنامه Hibernate خود در Eclipe . IDE ، مراحل زیر را به ترتیب انجام میدهیم:

1. یک پروژه جدید java درست کنید .

2. فایل های jarلازم جهت Hibernate را اضافه کنید.

3. کلاس Persistent Class را ایجاد نمایید.

4. تهیه فایل نقشه دهی mapping file برای کلاس. Persistent

5. ایجاد فایل پیکر بندی تنظیمات . configuration file

6. ایجاد کلاس های لازم و سپس تولید و نگهداری شی. Persistent object

7. اجرای برنامه.

مرحله اول ـ ایجاد پروژه جاوا:

از طریق پیمایش گزینه های File Menu - new - poroject - java porject یک پروژه جدید جاوا ایجاد کرده و نام آن برای مثال firsthbقرار دهید . سپس زدن دکمه های next - finish پنجره را ببندید.

مرحله دوم - اضافه کردن فایل های jar به Hibernate

برای اضافه نمودن فایل های jar مورد نظر به ویژه Hibernate مسیر زیر را طی کنید:

بر روی نام پروژه خود کلید راست نموده <- گزینه Build path -> Add eternal archives را بزنید . سپس کلید فایل های jar موجود را همانطور که در عکس زیر نشان داده شده ، انتخاب کرده و دکمه open را بزنید:

در مثال این درس ، ما برنامه خود را به پایگاه داده orcale متصل میکنیم . بنابراین بایستی فایل ajdelt. jar را اضافه کنید.

مرحله سوم ـ ایجاد کلاس persistent

در این مرحله قصد داریم تا یک کلاس persistent ، مشابه همانی که در درس قبل این درس ، درست کردیم را ایجاد کنیم . برای ایجاد کلاس Persistent class بر روی گزینه های class -> new -> کلیک نموده و سپس یک نام دلخواه را برای کلاس خود تعیین کنید (برای مثال. (com.tahlildadeh.my packge

در نهایت نیز گزینه finish را بزنید.

کد کلاس persistentبایستی به صورت زیر تعیین شود:

```
package com.javatpoint.mypackage;

public class Employee {
private int id;
private String firstName,lastName;
```

```
public int getId() {
    return id;
}
public void setId(int id) {
    this.id = id;
}
public String getFirstName() {
    return firstName;
}
public void setFirstName(String firstName) {
    this.firstName = firstName;
}
public String getLastName() {
    return lastName;
}
public void setLastName(String lastName) {
    this.lastName = lastName;
}


}
```

مرحله چهارم ـ ایجاد فایل mapping file جهت کلاس persistent

در این مرحله نیز ، یک mapping file مشابه انچه که در درس قبلی ایجاد کردیم ، درست خواهیم کرد . برای ایجاد mopping file مسیر file -> new -> src را طی کرده و سپس یک نام دخواه برای فایل خود تعیین کنید (برای مثال . (employee.hbm.xnl این فایل بایستی در خارج از package برنامه باشد.

کد فایل employee.hbm.xnl به صورت زیر است:

```
<!--?xml version='1.0' encoding='UTF-8'?-->


<hibernate-mapping>
 <class name="com.javatpoint.mypackage.Employee" table="emp1000">
   <id name="id">
    <generator class="assigned"></generator>
   </id>

   <property name="firstName"></property>
   <property name="lastName"></property>

 </class>

</hibernate-mapping>
```

مرحله پنجم ـ ایجاد فایل تنظیمات configuration file

فایل تنظیمات configuration شامل کلید اطلاعات مورد نیاز پایگاه داده از جمله connention _url ، driver _class، vsernam، passwordو... می باشد . از خاصیت hdm2ddl.anto property برای ایجاد جدول های پایگاه داده به صورت اتوماتیک ، استفاده می شود . در بخش های بعد ، به طور کامل راجع به کلاس Dialectتو ضیح خواهیم داد.

برای ایجاد فایل configoration بر روی گزینه scr کلیک راست کرده و سپس گزینه های file -> new را بزنید . در نهایت نیز یک نام دلخواه جهت فایل تنظیمات مثل hibernate. cfg.xnml به صورت زیر است:

کد:

```
<!--?xml version='1.0' encoding='UTF-8'?-->


<hibernate-configuration>

    <session-factory>
        <property name="hbm2ddl.auto">update</property>
        <property
name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
        <property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
        <property name="connection.username">system</property>
        <property name="connection.password">oracle</property>
        <property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
    <mapping resource="employee.hbm.xml">
    </mapping></session-factory>

</hibernate-configuration>
```

مرحله ششم -ایجاد کلاس لازم جهت دریافت و نگهداری شی persistent object

در کلاس زیر ، به صورت ساده شی employee object را در پایگاه داده نگهداری میکنیم.

کد:

```
package com.javatpoint.mypackage;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
```

```
public class StoreData {
public static void main(String[] args) {

    //creating configuration object
    Configuration cfg=new Configuration();
    cfg.configure("hibernate.cfg.xml");//populates the data of the
configuration file

    //creating seession factory object
    SessionFactory factory=cfg.buildSessionFactory();

    //creating session object
    Session session=factory.openSession();

    //creating transaction object
    Transaction t=session.beginTransaction();

    Employee e1=new Employee();
    e1.setId(115);
    e1.setFirstName("sonoo");
    e1.setLastName("jaiswal");

    session.persist(e1);//persisting the object

    t.commit();//transaction is committed
    session.close();

    System.out.println("successfully saved");

}
}
```

مرحله هفتم: اجرای برنامه Hiberate در Eclipse

قبل از اینکه بخواهید برنامه Hibernate را اجرا کنید ، مطمئن شوید که ساختار فایل ها را برنامه همانند عکس زیر باشد:

برای اجرای برنامه Hibernate، بر روی کلاس Store Data راست کلیک کرده و سپس دکمه Run As -> java Applicationرا کلیک کنید.

آموزش ایجاد برنامه Hibernate درMyEclipse

در این درس ، قصد داریم تا نحوه ایجاد و توسعه یک برنامهHibernate را در نرم افزار MyEclipse اموزش دهیم . برای ایجاد اولین برنامه Hibernate خود در IDE . MyEclipse ، مراحل زیر را به ترتیب انجام میدهیم:

1. یک پروژه جدید java درست کنید .
2. قابلیت های لازم جهت Hibernate را فعال کنید.
3. کلاس Persistent Class را ایجاد نمایید.
4. تهیه فایل نقشه دهی mapping file برای کلاس. Persistent
5. ایجاد فایل پیکر بندی تنظیمات configuration file و نگاشت و آدرس دهی فایل hbm در درون آن.
6. ایجاد کلاس های لازم و سپس تولید و نگهداری شی. Persistent object
7. فایل های jarلازم جهت Oracle را اضافه کنید.
8. اجرای برنامه.

مرحله اول ـ ایجاد پروژه جاوا:

با طی کردن مسیر زیر برنامه یک پروژه جدید جاوا را ایجاد کنید:

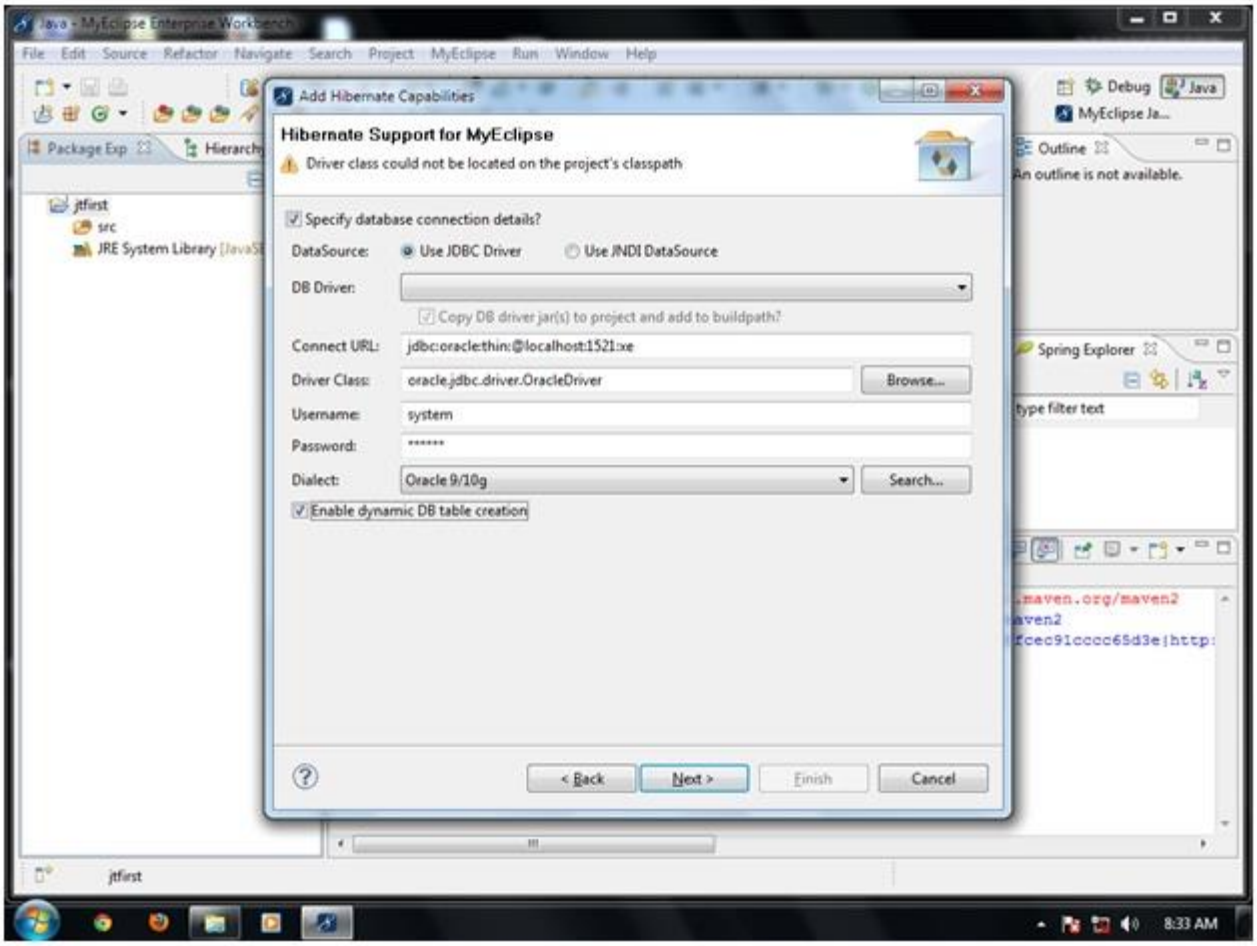
File Menu ◊ New ◊ project ◊ Java projectسپس یک نام دلخواه برای پروژه خود مثل firsthbتعیین کرده و با زدن دکمه های next ◊ finish پنجره را ببندید.

مرحله دوم ـ اضلفه کردن قابلیت های Hibernate به پروژه:

برای اضافه کردن فایل های jar به پروژه خود ، ابتدا آن را انتخاب کرده و سپس مسیر زیر را طی کنید:

برروی MyEclipse کلیک کرده --> add Hibernate capabilities --> Projection Capabilities --> next --> hext .

سپس تنظیمات لازم جهت اتصال به database را همانطور که در تصویر زیر نشان داده شده است ، تعیین کنید:



در تنظیمات فوق ، حتما تیک گزینه Enable dynamic table creation را فعال کرده تا برنامه جدول های پایگاه داده را به صورت اتوماتیک ، تولید کند.

پس دکمه next را زده و در مرحله بعد تیک گزینه Create Session Factorory class را بردارید ، زیرا قصد داریم تا کد لازم جهت گرفتن شی session object را خود به برای درک بهتر برنامه ، صورت دستی بنویسیم . سپس دکمه finish را بزنید.

در این محل ، فایل Configuration برنامه به صورت خودکار تولید می شود.

مرحله سوم ـ ایجاد کلاس Persistent

در این مرحله نیز یک کلاس Persistent ، همانند آنچه در درس قبل ایجاد کردیم ، درست می کنیم . برای این منظور برروی گزینه src کلیک راست کرده ، سپس دکمه های New و Class را به ترتیب انتخاب کنید . در آخر یک نام دلخواه جهت کلاس خود مثل com.tahlildadeh.mypackage تعیین کرده و گزینه finish را بزنید.

کد کلاس Persiitent در فایل Employee.java به صورت زیر خواهد بود:

```
package com.javatpoint.mypackage;

public class Employee {
private int id;
private String firstName,lastName;

public int getId() {
    return id;
}
public void setId(int id) {
    this.id = id;
}
public String getFirstName() {
    return firstName;
}
public void setFirstName(String firstName) {
    this.firstName = firstName;
}
public String getLastName() {
    return lastName;
}
public void setLastName(String lastName) {
    this.lastName = lastName;
}


}
```

مرحله چهارم ـ ایجاد فایل نقشه دهی mapping file جهت کلاس Persistent

در این مرحله نیز ، همانند آنچه در درس قبل انجام دادیم یک mapping file جهت پروژه ایجاد می کنیم . برای ایجاد یک mapping file ، برروی گزینه src کلیک راست کرده و سپس گزینه های New و file را به ترتیب انتخاب کنید . سپس یک نام دلخواه برای فایل خود مثل hibernate,hbm.xml تعیین نموده و توجه کنید این فایل بایستی بیرون از package برنامه باشد.

کد زیر را درون فایل mapping خود قرار دهید:

```
<!--?xml version='1.0' encoding='UTF-8'?-->


<hibernate-mapping>
 <class name="com.javatpoint.mypackage.Employee" table="emp1000">
   <id name="id">
    <generator class="assigned"></generator>
   </id>
```

```
   <property name="firstName"></property>
   <property name="lastName"></property>

 </class>

</hibernate-mapping>
```

مرحله پنجم ـ ایجاد یک مسیر جهت فایل mapping در فایل تنظیمات Configuration

فایل تنظیمات برنامه (hibernate.cfg.xml) را باز کرده و یک مسیر جهت فایل mapping برنامه ، به صورت زیر در آن ایجاد کنید:

کد:

```
<mapping resource="employee.hbm.xml">
</mapping>
```

اکنون فایل Configuration به صورت زیر خواهد شد:

کد:

```
        <!--?xml version='1.0' encoding='UTF-8'?-->


<hibernate-configuration>

    <session-factory>
        <property name="hbm2ddl.auto">update</property>
        <property
name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
        <property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
        <property name="connection.username">system</property>
        <property name="connection.password">oracle</property>
        <property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
    <mapping resource="employee.hbm.xml">
    </mapping></session-factory>

</hibernate-configuration>
```

مرحله ششم ـ ایجاد کلاس لازم جهت دریافت و نگهداری شی persistent object

:

در کلاس زیر ، به سادگی شی employee object را برای database ذخیره می کنیم:

کد:

```
package com.javatpoint.mypackage;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;

public class StoreData {
public static void main(String[] args) {

    //creating configuration object
    Configuration cfg=new Configuration();
    cfg.configure("hibernate.cfg.xml");//populates the data of the
configuration file

    //creating seession factory object
    SessionFactory factory=cfg.buildSessionFactory();

    //creating session object
    Session session=factory.openSession();

    //creating transaction object
    Transaction t=session.beginTransaction();

    Employee e1=new Employee();
    e1.setId(115);
    e1.setFirstName("sonoo");
    e1.setLastName("jaiswal");

    session.persist(e1);//persisting the object

    t.commit();//transaction is commited
    session.close();

    System.out.println("successfully saved");

}
}
```

مرحله هفتم ـ اضافه کردن فایل jar لازم جهت oracle

برای اضافه کردن فایل jar لازم برای) oracle به نام (ojdbct.jar ، برروی پروژه کلیک راست نموده و به

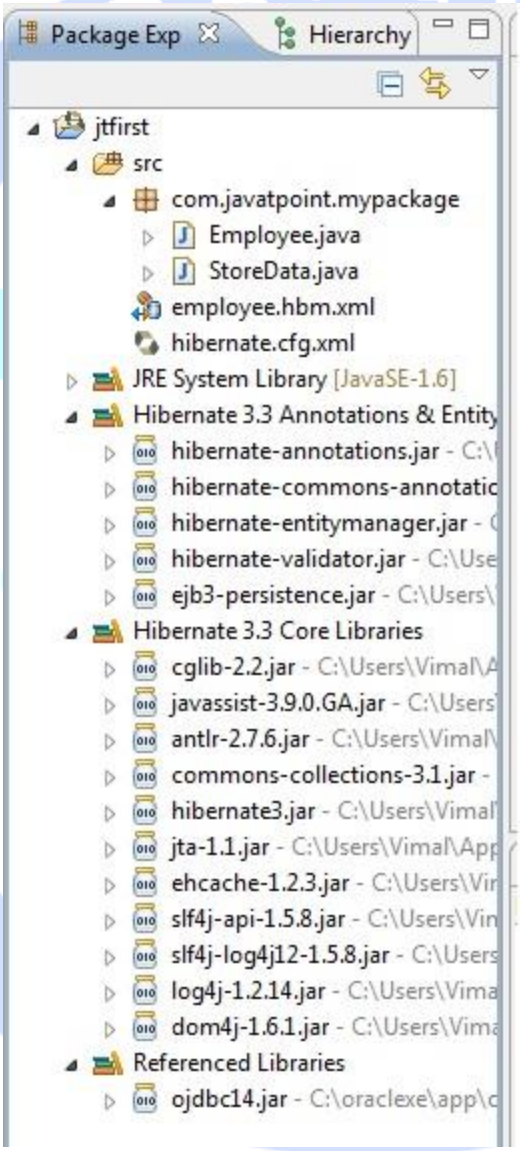
ترتیب مراحل زیر را طی کنید:

Build path ◊ add external archives

سپس فایل jdbct.jar را انتخاب کرده و دکمه open را بزنید.

مرحله آخر ـ اجرای برنامه Hibernate

قبل از اجرای برنامه Hibernate ، مطمئن شوید ساختار فایل های آن همانند تصویر زیر باشد:



جهت اجرای برنامه Hibernate ، برروی کلاس Store Data کلیک راست کرده و سپس دکمه های Run As و Java Applicationرا انتخاب کنید.

درس ششم : آموزش استفاده از annonation در Hibernate

برنامه های Hibernate را می توان به همراه annotation ها نیز ایجاد کرد . همانطور که می دانید ، annotation یا به معنی فارسی " توضیحات متنی درباره داده " ، اطلاعاتی است که در لا به لای کدهای برنامه قرار داده می شوند . این کدها توضیحاتی را به نحوه عملکرد برنامه و روند اجرای آن ارایه می کنند که هم برای کاربر و هم برای کامپایلر جاوا قابل فهم است ، ولی انرژی برروی کدهای برنامه ندارد و در خروجی هم نمایش داده نمی شود.

Annotationهایی زیادی از جمله @Entity ، @Id ، @Table و ... وجود دارد که از آنها می توانید در برنامه های Hibernate استفاده کنید.

Annotationها در Hibernateبرپایه JPA 2 تعریف شده اند و قابلیت های آن را دارا می باشند . کلید JPA Annotationها در پکیج javax.persistence.* تعریف شده اند Entity Manger . در Hibernate ، رابط کاربری و Life cycle تعیین شده توسط JPA را اجرا و تولید می کند.

مهمترین فایده استفاده از annotation در Hibernate، این است که دیگر نیازی ندارد تا فایل (mapping hbm) را برای برنامه ایجاد کنید . در اینجا Hibernate annotation برای تولید meta data به کار می رود.

مثال عملی ایجاد یک برنامه Hibernate به همراه Annotation

برای ایجاد یک برنامه Hibernate به همراه annotation ، بایستی 4 مرحله زیر را انجام دهید:

1. فایل های jar لازم جهت) oracle اگر پایگاه داده برنامه oracle باشد (و annotation را به برنامه اضافه کنید.
2. کلاس Persistent Class را ایجاد نمایید.
3. mapping (آدرس دهی) لازم را جهت کلاس Persistent در فایل تنظیمات Configuration برنامه ایجاد کنید.
4. کلاس لازم جهت ایجاد و نگهداری Persistent را تعیین و ایجاد کنید.

در ادامه به تشریح کامل هر یک از مراحل فوق خواهیم پرداخت.

مرحله اول_اضافه کردن فایل های jar لازم برای oracle و annotation

برای annotation هم بایستی فایل های زیر را به برنامه اضافه کنید:

- hibernate-commons-annotations.jar
- ejb3-persistence.jar
- hibernate-annotations.jar

مرحله دوم_آموزش ایجاد کلاس Persistent Class

در این مرحله ، یک کلاس persistent همانند آنچه در مرحله قبل ایجاد کردیم ، به پروژه اضافه می کنیم . با این تفاوت که در کلاس برنامه جدید ، از annotation های استفاده می کنیم.

@Entity کلاس جاری را به عنوان یک Entity مشخص می کند.

@Table نام جدولی که قرار است اطلاعات Entity فوق در آن قرار بگیرد را تعیین می کند . اگر از @Table annotation استفاده نکنید ، برنامه به صورت پیش فرض ، نام کلاس را به عنوان نام جدول تعیین می کنید.

@Id ، شناسه (Identifier) لازم را جهت Entity تعیین می کند @column . ، اطلاعات لازم برای column هر property یا فیلد در برنامه را تعیین می کند . اگر از @column annotation استفاده نشود ، برنامه نام هر property را به صورت پیش فرض برای هر column آن در نظر می گیرد.

کد زیر ، محتویات فایل Employee.java را نشان داده که نحوه استفاده از annotation ها در آن مشخص شده است :

```
package com.javatpoint;

import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name= "emp500")
public class Employee {
@Id
private int id;
private String firstName,lastName;

public int getId() {
    return id;
}
public void setId(int id) {
```

```
    this.id = id;
}
public String getFirstName() {
    return firstName;
}
public void setFirstName(String firstName) {
    this.firstName = firstName;
}
public String getLastName() {
    return lastName;
}
public void setLastName(String lastName) {
    this.lastName = lastName;
}


}
```

مرحله سوم_ اضافه کردن mapping جهت کلاس Persistent در فایل Configuration

فایل تنظیمات برنامه hibernate.cgf.xml را باز کرده و کد خط زیر را برای آدرس دهی (mapping) کلاس Persistent به آن اضافه کنید:

```
<mapping class="com.javatpoint.Employee">   </mapping>
```

اکنون فایل تنظیمات Configuration به صورت زیر تغییر خواهد کرد:

```
<!--?xml version='1.0' encoding='UTF-8'?-->


<hibernate-configuration>

<session-factory>
 <property name="hbm2ddl.auto">create</property>
 <property name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
 <property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
 <property name="connection.username">system</property>
 <property name="connection.password">oracle</property>
 <property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>

 <mapping class="com.javatpoint.Employee">
 </mapping></session-factory>

</hibernate-configuration>
```

مرحله چهارم_ ایجاد کلاس لازم جهت تولید و نگهداری شی persistent object

در این کلاس ، قصد داریم تا شی employee object را در پایگاه داده نگهداری کنیم . در این بخش ، از کلاس Annotation Configuration برای دریافت اطلاعات لازم جهت آدرس دهی (mapping) از کلاس Persistentاستفاده می کنیم . همانند کد زیر:

```
package com.javatpoint.mypackage;

package com.javatpoint;

import org.hibernate.*;
import org.hibernate.cfg.*;

public class Test {
public static void main(String[] args) {
    Session session=new AnnotationConfiguration()
         .configure().buildSessionFactory().openSession();

    Transaction t=session.beginTransaction();

    Employee e1=new Employee();
    e1.setId(1001);
    e1.setFirstName("sonoo");
    e1.setLastName("jaiswal");

    Employee e2=new Employee();
    e2.setId(1002);
    e2.setFirstName("vimal");
    e2.setLastName("jaiswal");

    session.persist(e1);
    session.persist(e2);

    t.commit();
    session.close();
    System.out.println("successfully saved");
}
}
```

طراحی برنامه تحت وب با Hibernate

در این درس قصد داریم تا نحوه طراحی یک برنامه تحت وب Web Applicationبه استفاده از Hibernateرا آموزش دهیم . جهت ایجاد برنامه تحت وب از JSP به طراحی لایه peresentation logic ، از کلاس Bean برای نگهداری کردن داده ها و از کلاس های DAO جهت تعیین کد های پایگاه داده استفاده خواهیم کرد.

همانطور که در درس های قبل ، یک برنامه ساده Hibernate نیز ، نیاز به انجام کارهای زیادی نیست . در همچنین برنامه ای ، ما داده ها و اطلاعات را به وسیله یک فایل JPS از کاربرد دریافت می کنیم.

مثال عملی ایجاد یک برنامه تحت وب با Hibernate

در مثال عملی برنامه تحت وب Hibernate، قصد داریم تا با استفاده از یک فرم ساده ثبت نام ، اطلاعات کاربرد را در پایگاه داده برنامه ذخیره کنیم.

کد زیر ، محتویات فایل index.jsp یا صفحه اصلی برنامه را نشان می دهد. در این فایل ، فرم ساده ثبت نام ، اطلاعات کاربر دریافت کرده و به وسیله متد postانها را به صفحه register.jsp ارسال می کند.

```
< form action="register.jsp" method="post" >
  Password:<input name="password" type="password"><br><br>
  Email ID:<input name="email" type="text"><br><br>
  <input value="register" type="submit">"
< /form >
```

کد زیر مربوط به فایل register.jsp را به صورت پارامتر دریافت کرده و انها را به عنوان یک object از کلاس userنگهداری می کند سپس ، متد registerرا فراخوانی کرده و با استفاده از کلاس UserDao و محتویات شی user objectرا جهت ثبت در پایگاه داده ، پاس می دهد.

```
<%@page import="com.javatpoint.mypack.UserDao"%>
<jsp:usebean id="obj" class="com.javatpoint.mypack.User">
</jsp:usebean>
<jsp:setproperty property="*" name="obj">

<%
int i=UserDao.register(obj);
if(i>0)
out.print("You are successfully registered");

%>
</jsp:setproperty>
```

کد زیر نیز ، کد کلاس Bean را که نماینده کلاس persistent در Hibernate است را نشان می دهد :

```
package com.javatpoint.mypack;

public class User {
private int id;
private String name,password,email;

//getters and setters

}
```

از طریق فایل زیر ، اطلاعات شی User object را به جدول متناظر آن در پایگاه داده مسیر دهی mapping می کنیم :

```
<!--?xml version='1.0' encoding='UTF-8'?-->


 <hibernate-mapping>
 <class name="com.javatpoint.mypack.User" table="u400">
 <id name="id">
 <generator class="increment"></generator>
 </id>
 <property name="name"></property>
 <property name="password"></property>
 <property name="email"></property>
 </class>

 </hibernate-mapping>
```

کلاس Dao زیر نیز ، حاوی متد های لازم جهت ذخیره هر intance از کلاس User می باشد :

```
package com.javatpoint.mypack;


import org.hibernate.Session;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;

public class UserDao {

public static int register(User u){
 int i=0;
 Session session=new Configuration().
  configure().buildSessionFactory().openSession();

  Transaction t=session.beginTransaction();
  t.begin();

  i=(Integer)session.save(u);

  t.commit();
  session.close();

  return i;
 }
}
```

فایل Configoration به نام hibernate.cfg. xml نیز حاوی اطلاعات مورد نیاز پایگاه داده و فایل نقشه دهی برنامه mapping file

```
<!--?xml version='1.0' encoding='UTF-8'?-->
```

```
<hibernate-configuration>

<session-factory>
 <property name="hbm2ddl.auto">create</property>
 <property name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
 <property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
 <property name="connection.username">system</property>
 <property name="connection.password">oracle</property>
 <property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>

 <mapping resource="user.hbm.xml">
 </mapping></session-factory>

</hibernate-configuration>
```

آموزش کار با کلاس های سازنده Hibernate Generator Class

زیر عنصر المنت id ، جهت تولید شناسه های منحصر به فرد (unique identifier) برای اشیای کلاس Persistent به کار میرود . کلاس های سازنده (generat class) زیادی درچهارچوب کاری Hibernate تعیین شده اند که از آنها می توانید برای تولید سریع کد برنامه استفاده کنید.

کلیه کلاس های generator ، رابطه کاربری org.hibernate.id.Identifier Generator را اجرا می کنند . همچنین ، برنامه نویس می تواند کلاس های سازنده دلخواه خود را با استفاده از رابطه رابط کاربری Identifier Generator ایجاد کند . چهارچوب کاری Hibernate ، کلاس های سازنده درون ساخته زیادی دارد که از آن می تواند به موارد زیر اشاره کرد:

- assigned
- increment
- sequence
- hilo
- native
- identity

- seqhilo
- uuid
- guid
- select
- foreign
- sequence-identity

در ادامه به تشریح و آموزش هریک از کلاس های سازنده فوق خواهیم پرداخت.

1. کلاس: assigned

درصورتی که کلاس generator برای برنامه تعیین نشده باشد ، این کلاس به صورت پیش فرض برای المنت فعال خواهد شد . در این حالت ، خود برنامه id را تعیین می کند . مثل کد زیر:

```
....
 <hibernate-mapping>
  <class ...="">
    <id ...="">
     <generator class="assigned"></generator>
    </id>

....

  </class>
 </hibernate-mapping>
```

2. کلاس: increment

این کلاس در صورتی که هیچ پروسه ای دیگر ، اطلاعات به جدول وارد نکنند ، idهای منحصر به فرد را تولید خواهد کرد . کلاس increment، برای انواع داده ای short ، int، long، شناسه id تولید می کند . معمولا اولی که تولید می شود 1 بوده و یا واحد 1 افزایش پیدا می کند.

کد تعریف کلاس increment به صورت زیر است:

```
....
<hibernate-mapping>
 <class ...="">
   <id ...="">
    <generator class="increment">></generator>
   </id>
```

```
   ....

 </class>
</hibernate-mapping>
```

3. کلاس: sequence

این کلاس ، از ترتیب sequence پایگاه داده برای تولید شناسه های id استفاده می کند . اگر sequence در دیتابیس تعیین نشده باشد ، این کلاس به صورت اتوماتیک یک sequence تولید می کند.

برای مثال در پایگاه داده oracle ، کلاس sequence یک ترتیب به نام HIBERNATE SEQUENCE ایجاد می کند . این کلاس در پایگاه ها داده oracle ، DB2 ، SAP DB ، Mckoi و Postgre SQL از sequence استفاده می کند ، اما در دیتابیس interbase از generator استفاده می کند . ساختار دستوری کلاس sequence به صورت زیر است:

```
.....
<id ...="">
<generator class="sequence">
</generator>
</id>
.....
```

برای ایجاد sequence دلخواه خود ، از زیر عنصر param المنت به صورت زیر استفاده کنید:

```
.....
<id ...="">
 <generator class="sequence">
     <param name="sequence">your_sequence_name
 </generator>
</id>
.....
```

4. کلاس hilo

کلاس hilo ، از الگوریتم (high and low شناسه با بیشترین و کمترین عدد) برای ایجاد id جهت انواع داده ای short ، int ، long ، استفاده می کند . ساختار دستوری استفاده از کلاس hilo به تشریح زیر است:

```
.....
<id ...="">
 <generator class="hilo">
</id>
.....
```

5. کلاس native

کلاس native ، برحسب database ای که در برنامه استفاده می شود ، از استراتژی های identify ، sequenceیا hilo استفاده خواهد کرد . ساختار دستوری استفاده از کلاس native به صورت زیر است:

```
.....
<id ...="">
 <generator class="native"></generator>
</id>
.....
```

6. کلاس identify

کلاس identify در پایگاه های داده Sybase ، My SQL ، MS SQL ، DB2 و Hypersonic SQL جهت پشتیبانی از id column استفاده می شود.

شناسه (id) بازگرداننده توسط کلاس identify از انواع int ، short یا long خواهد بود.

7. کلاس: seghilo

کلاس seqhilo از الگوریتم high and low برای sequence تعیین شده ، استفاده می کند id .

برگشتی این کلاس از نوع int ، short یا long خواهد بود.

8. کلاس uvid

کلاس uvid از الگوریتم 128 بیسی UVID برای تولید id ها ، استفاده می کند id . بازگردانده شده از این کلاس از انواع string بوده و به دلیل استفاده از Ip ، در سطح کل شبکه منحصر به فرد خواهد بود ، شناسه های UVID در فرمت هگزادمیکال و با طول 32 تعریف میشوند.

9. کلاس guid

کلاس guid از GUID توسط دیتابیس که از نوع string باشد برای id ها استفاده می کند . کلاس guid، برروی پایگاه های داده MS SQL و MySQL کار میکند.

10. کلاس select

کلاس select از کلید اصلی (primary key) که توسط رویدادهای پایگاه داده بازگردانده می شود ، برای تولید id استفاده می کند.

11. کلاس foreign

کلاس foreign از object دیگری که مرتبط با object جاری بوده و با استفاده از ارتباط ، اقدام به تولید id می کند.

12. کلاس sequence-identify از یک استراتژی ترتیبی (sequence) خاص برای تولید id ها استفاده می کند . این کلاس فقط برروی پلتفرم oracle log پشتیبانی می شود.

درس نهم آموزش SQL Dialect درHibernate

برای اتصال هر برنامه hibernate به پایگاه داده ، شما بایستی SQL Dialect متناسب را برای آن تعیین کنید .

اما SQL Dialect چیست. در معنی لغوی)) گویش یا لحجه ((SQL ترجمه میشود.

سیتم های داده ای مختلف از برخی دستورات یا Sytax های مخصوص به خود در SQLاستفاده می کنند .

بنابراین شما در برنامه خود بایستی تعیین کنید که می خواهید به چه دیتابیسی وصل شوید ، تا برنامه بهتر بتواند زبان دستورات را بفهمند.

کلاس های زیادی برای SQLDialect در پکیج org.hibernate.dialect تعیین شده اند که لیست آنها عبارتست از:

RDBMS	Dialect
Oracle (any version)	org.hibernate.dialect.OracleDialect
Oracle۹i	org.hibernate.dialect.Oracle۹iDialect
Oracle۱۰g	org.hibernate.dialect.Oracle۱۰gDialect
MySQL	org.hibernate.dialect.MySQLDialect
MySQL with InnoDB	org.hibernate.dialect.MySQLInnoDBDialect
MySQL with MyISAM	org.hibernate.dialect.MySQLMyISAMDialect
DB۲	org.hibernate.dialect.DB۲Dialect
DB۲ AS/۴۰۰	org.hibernate.dialect.DB۲۴۰۰Dialect
DB۲ OS۳۹۰	org.hibernate.dialect.DB۲۳۹۰Dialect
Microsoft SQL Server	org.hibernate.dialect.SQLServerDialect
Sybase	org.hibernate.dialect.SybaseDialect
Sybase Anywhere	org.hibernate.dialect.SybaseAnywhereDialect
PostgreSQL	org.hibernate.dialect.PostgreSQLDialect
SAP DB	org.hibernate.dialect.SAPDBDialect
Informix	org.hibernate.dialect.InformixDialect
HypersonicSQL	org.hibernate.dialect.HSQLDialect
Ingres	org.hibernate.dialect.IngresDialect
Progress	org.hibernate.dialect.ProgressDialect
Mckoi SQL	org.hibernate.dialect.MckoiDialect
Interbase	org.hibernate.dialect.InterbaseDialect
Pointbase	org.hibernate.dialect.PointbaseDialect
FrontBase	org.hibernate.dialect.FrontbaseDialect
Firebird	org.hibernate.dialect.FirebirdDialect

درس دهم آموزش Hibernate Logging با Log4j و فایل xml در Hibernate

قابلیت Logging در Hibernate ، به برنامه نویس امکان نوشتن جزئیات رویدادهای برنامه log detalis)) را به صورت مداوم در یک فایل می دهد.

از چهار چوب های کاری آماده Log4j و Logback در Hibernate برای پشتیبانی و انجام امور logging استفاده می شد دو راه برای انجام logging به وسیله log4j وجود دارد که عبارتند از:

- وسیله استفاده از یک فایل xml به نام. log 4j. xml
- به وسیله یک فایل log4j. properties

مراحل انجام عملیات loging به وسیله log4j با فایل xml :

1. فایل log4j.jar را به وسیله Hibernate لود کنید.

2. فایل log4j.xml را درون پوشه src ایجاد کنید (این فایل بایستی موازی و هم پوشه فایل hibenate.cfg.xnl باشد)

مثال عملی انجام عملیات logging با استفاده از log4j و فایل xml :

شما می توانید قابلیت logging در Hibernate را با انجام در هر مرحله ساده در هر برنامه Hibernate های فعال کنید.

مرحله اول : لود کردن فایل های jar لازم در ابتدا بایستی فایل های jar مورد نیاز عملیات loging به نام های log4j به نام های log4j.jar و slf4j.jar را به پروژه اضافه کنید.

مرحله دوم : ایجاد فایل log4j.xml در پروژه.

در مرحله دوم بایستی فایل log4j.xml را در پروژه ایجاد کنید . در مثال این درس ، کلیه اطلاعات log برنامه در فایل c:/javatpoing.log ذخیره خواهد شد.

کد فایل log4j.xml به صورت زیر است:

```
<!--?xml version="1.0" encoding="UTF-8"?-->

<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/"
debug="false">
<appender name="CONSOLE" class="org.apache.log4j.ConsoleAppender">
 <layout class="org.apache.log4j.PatternLayout">
  <param name="ConversionPattern" value="[%d{dd/MM/yy hh:mm:ss:sss z}] %5p
%c{2}: %m%n">
 </layout>
</appender>
    <appender name="ASYNC" class="org.apache.log4j.AsyncAppender">
        <appender-ref ref="CONSOLE">
        <appender-ref ref="FILE">
</appender-ref></appender-ref></appender>
<appender name="FILE" class="org.apache.log4j.RollingFileAppender">
    <param name="File" value="C:/javatpointlog.log">
    <param name="MaxBackupIndex" value="100">
 <layout class="org.apache.log4j.PatternLayout">
  <param name="ConversionPattern" value="[%d{dd/MM/yy hh:mm:ss:sss z}] %5p
%c{2}: %m%n">
</layout>
</appender>
    <category name="org.hibernate">
        <priority value="DEBUG">
    </priority></category>
```

```
    <category name="java.sql">
        <priority value="debug">
    </priority></category>
    <root>
        <priority value="INFO">
        <appender-ref ref="FILE">
    </appender-ref></priority></root>
</log4j:configuration>
```

آموزش Hibernate در Logging با Log4j و فایل properties

همانطور که در درس قبل تشریح کردیم ، چهار چوب های کاری آماده log4j و logbackبرای انجام عملیات loggingبه وسیله log4j در Hibernate وجود دارد که عبارتند از:

1. با استفاده از یک فایل xml به نام. log4j.xml
2. با استفاده از یک فایل به نام. log4j.properties

در این درس، قصد داریم تا نحوه انجام عمل logging با استفاده از یک فایل properties را آموزش دهیم .

مراحل انجام عملیات logging با log4j و فایل propertiesدر Hibernate

برای انجام عملیات logging در Hibernate با استفاده از فایل properties بایستی دو مرحله زیر را انجام دهید:

1. لود فایل هایjar لازم جهت log4j در. Hibernate
2. ایجاد فایل log4j.properties درون پوشه src (برنامه این فایل بایستی موازی و هم پوشه فایل hibernate.cfg.xmlباشد)

مثال عملی انجام عمل logging با استفاده از log4j و فایل ایل propertiesدر Hibernate

به وسیله انجام دو مرحله ساده زیر ، میتوانید قابلیت logging را در هر برنامه Hibernate ای به سادگی فعال نمایید:

- لود فایل های لازم jar به پروژه:

در هر مرحله بایستی دو فایل jar به نام های log4j.jar و slf4j.jarرا به پروژه اضافه کنید.

<--مسیر دانلود فایل های jar لازم

- ایجاد فایل log4j .properties در برنامه:

در مرحله دوم بایستی فایل log4j.properties را در پوشه c:\\javapoint hibernate.logایجاد کنید. کد فایل properties به صورت زیر است . تمامی اطلاعات log برنامه در این فایل ذخیره می شود.

```
# Direct log messages to a log file
log4j.appender.file=org.apache.log4j.RollingFileAppender
log4j.appender.file.File=C:\\javatpointhibernate.log
log4j.appender.file.MaxFileSize=1MB
log4j.appender.file.MaxBackupIndex=1
log4j.appender.file.layout=org.apache.log4j.PatternLayout
log4j.appender.file.layout.ConversionPattern=%d{ABSOLUTE} %5p %c{1}:%L - %m%n

# Direct log messages to stdout
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.Target=System.out
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{ABSOLUTE} %5p %c{1}:%L -
%m%n

# Root logger option
log4j.rootLogger=INFO, file, stdout

# Log everything. Good for troubleshooting
log4j.logger.org.hibernate=INFO

# Log all JDBC parameters
log4j.logger.org.hibernate.type=ALL
```

آموزش آدرس دهی (mapping) درHibernate

در Hibernate ، می توانید سلسله مراتب کلاس های برنامه را در یک جدول پایگاه داده نگاشت یا آدرس دهی (map) کنید . سه نوع استراتژی برای این منظور در Hibernate وجود دارد:

1. جدول برمبنای سلسله مراتب(Table Per Hierarchy)
2. جدول برمبنای کلاس های واقعی(Table Per Concerete Class)

3. جدول برمبنای کلاس های زیرمجموعه(Table Per Subclass)

در این درس به معرفی اجمالی هریک از استراتژی های فوق می پردازیم ، پس در درس های بعدی به صورت کامل آنها را تشریح خواهیم کرد.

جدول برمبنای سلسله مراتب (Table Per Hierarchy)

در مدل جدول برمبنای سلسله مراتب (Table Per Hierarchy) ، یک جدول تنها برای نگاشت و آدرس دهی (map) کل سلسله مراتب کلاس ها لازم است . همچنین ، این جدول یک ستون اضافه به نام ستون تفکیک کننده (discriminator column) دارد که برای شناسایی و تمایز کلاس از هم به کار خواهد رفت . از طرف دیگر ، مقادیر خالی (Nullable) نیز در این جدول نگهداری خواهند شد.

- آموزش نگاشت سلسله مراتب (Table Per Hirarechy) با استفاده از فایل. xml
- آموزش نگاشت سلسله مراتب (Table Per Hirarechy) با استفاده از . Annotation

جدول برمبنای کلاس واقعی (Table Per Concerete Class)

در مدل جدول برمبنای کلاس های مادر ، به ازای هرکلاس یک جدول شاخته می شود . اما ستون های مشابه و تکراری در جدول فرعی ذخیره می شوند.

- آموزش نگاشت سلسله مراتب(Table Per Concerete Class) به وسیله یک فایل. xml
- آموزش نگاشت سلسله مراتب (Table Per Concerete Class) به وسیله. Annotation

جدول برمبنای کلاس های فرعی (Table Per Subclass)

در مدل جدول برمبنای کلاس های فرعی نیز ، به ازای هرکلاس یک جدول ساخته می شود اما براساس کلید خارجی جدول (forigen) بنابراین در این مدل ستون تکراری نخواهیم داشت.

- آموزش نگاشت سلسله مراتب (Table Per Subclass) با استفاده از فایل. xml
- آموزش نگاشت سلسله مراتب (Table Per Subclass) با استفاده از. Annotation

```
<relativelayout xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent" android:layout_height="match_parent"
android:paddingbottom="@dimen/activity_vertical_margin"
android:paddingleft="@dimen/activity_horizontal_margin"
android:paddingright="@dimen/activity_horizontal_margin"
android:paddingtop="@dimen/activity_vertical_margin"
tools:context=".MainActivity">
```

```
    <edittext android:id="@+id/main_input"
android:layout_width="wrap_content" android:layout_height="wrap_content"
android:layout_alignparenttop="true"
android:layout_alignparentstart="true"
android:layout_alignparentend="true">
    </edittext>



    <button android:id="@+id/button1" android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_alignleft="@+id/main_input"
android:layout_below="@+id/main_input" android:layout_margintop="31dp"
android:onclick="onClick" android:text="Start">
    </button>



</relativelayout>
```

آموزش مدل نگاشت Table Per Hierachy به وسیله فایل xml

به وسیله این استراتژی، می توانید کل سلسله مراتب (Hierachy) برنامه را در یک جدول تنها، نگاشت یا آدرس دهی (map) کنید. در این مدل، یک ستون اضافه در جدول وجود دارد که به آن ستون تفکیک کننده یا discriminator column می گویند و برای تشخیص و تمایز کلاس ها از یکدیگر به کار می رود.

بیایید ابتدا نگاهی به صورت مسئله بیاندازیم. می خواهیم کل سلسه مراتب یا Hierachy که در دیاگرام زیر نمایش داده شده است را در یک جدول پایگاه داده نگاشت (map) کنیم.

سه کلاس در سلسله مراتب فوق وجود دارد. کلاس Employee کلاس مادر یا کلاس super classدو کلاس دیگر یعنی کلاس Coutract_Employee و کلاس .Regular_Employeeفایل آدرس دهی (mapping) به کار رفته جهت این سلسله مراتب دارای کد زیر است:

```
<!--?xml version='1.0' encoding='UTF-8'?-->


<hibernate-mapping>
<class name="com.javatpoint.mypackage.Employee" table="emp121" discriminator-
value="emp">
<id name="id">
<generator class="increment"></generator>
</id>
```

```
<discriminator column="type" type="string"></discriminator>
<property name="name"></property>

<subclass name="com.javatpoint.mypackage.Regular_Employee" discriminator-
value="reg_emp">
<property name="salary"></property>
<property name="bonus"></property>
</subclass>

<subclass name="com.javatpoint.mypackage.Contract_Employee" discriminator-
value="con_emp">
<property name="pay_per_hour"></property>
<property name="contract_duration"></property>
</subclass>

</class>

</hibernate-mapping>
```

در مدل Table Per Hierachy یک ستون اضافه به عنوان ستون تفکیک کننده یا discriminator Column به چهارچوب کاری Hiberante اضافه می شود که نوع یا مشخصات رکورد جاری را ثبت می کند. این ستون، در اصل برای متمایز کردن و امکان تشخیص رکوردها از هم به کار می رود. برای این منظور بایستی زیرالمنت discriminatorدر کد کلاس تعیین شود.

از طرف دیگر، زیرالمنتSubclass ، کلاس زیرمجموعه یا وابسته Subclass را مشخص می کند. در این مثال، کلاس های Regular_Employee و Contract_Employee کلاس های وابسته یا Subclass کلاس اصلی یعنی Employee هستند.

ساختار جدول مورد استفاده جهت سلسله مراتب Hierarchy به صورت زیر است:

مثال عملی کار با مدل نگاشتTable Per Hierarchy

در این مثال، ما 3 کلاس مورد نیاز برنامه را تولید کرده و سپس اطلاعات آدرس دهی (mapping) آن ها را در فایل employee.bhm.xml قرار می دهیم. برای این منظور مراحل زیر را انجام دهید:

1. (1ایجاد کلاس های: Persistent Class

در مرحله اول بایستی Persistent Class را ایجاد کرده که وراثت را در برنامه مشخص می کند. کد زیر، نحوه تعریف 3 کلاس اصلی برنامه را نشان داده است:

File: Employee.java

```
package com.javatpoint.mypackage;

public class Employee {
private int id;
private String name;

//getters and setters
}
```

File: Regular_Employee.java

```
package com.javatpoint.mypackage;

public class Regular_Employee extends Employee{
private float salary;
private int bonus;

//getters and setters
}
```

File: Contract_Employee.java

```
package com.javatpoint.mypackage;

public class Contract_Employee extends Employee{
    private float pay_per_hour;
    private String contract_duration;

//getters and setters
}
```

2. ایجاد فایل آدرس دهی لازم جهت کلاس Persistent Class

کد فایل employee.bhm.xml که آدرس دهی (mapping) سلسله مراتب فوق را توضیح می دهد، به صورت زیر است:

```
<!--?xml version='1.0' encoding='UTF-8'?-->


<hibernate-mapping>
<class name="com.javatpoint.mypackage.Employee" table="emp121" discriminator-
value="emp">
<id name="id">
<generator class="increment"></generator>
</id>
```

```
<discriminator column="type" type="string"></discriminator>
<property name="name"></property>

<subclass name="com.javatpoint.mypackage.Regular_Employee" discriminator-
value="reg_emp">
<property name="salary"></property>
<property name="bonus"></property>
</subclass>

<subclass name="com.javatpoint.mypackage.Contract_Employee" discriminator-
value="con_emp">
<property name="pay_per_hour"></property>
<property name="contract_duration"></property>
</subclass>

</class>

</hibernate-mapping>
```

3. اضافه کردن آدرس فایل hbm در فایل تنظیمات: Configuration

کد فایل hibernate.cgf.xml را باز کرده و المنت زیر را جهت آدرس دهی (mapping) فایل hbm به آن اضافه کنید:

```
<mapping resource="employee.hbm.xml">
</mapping>
```

پس از اصلاح، کد فایل بایستی به صورت زیر تغییر کند:

```
<!--?xml version='1.0' encoding='UTF-8'?-->


<hibernate-configuration>

    <session-factory>
        <property name="hbm2ddl.auto">update</property>
        <property
name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
        <property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
        <property name="connection.username">system</property>
        <property name="connection.password">oracle</property>
        <property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
    <mapping resource="employee.hbm.xml">
    </mapping></session-factory>

</hibernate-configuration>
```

4. ایجاد کلاس لازم جهت نگهداری شی Persistent

در کلاس زیر که کد آن نمایش داده شده است، یک کلاس به نام ta و StoreD ایجاد می کنیم تا شی Persistent Objectرا در سطح برنامه نگهداری کند:

File: StoreData.java

```
package com.javatpoint.mypackage;
import org.hibernate.*;
import org.hibernate.cfg.*;

public class StoreData {
public static void main(String[] args) {
    Session session=new Configuration().configure("hibernate.cfg.xml")
                        .buildSessionFactory().openSession();

    Transaction t=session.beginTransaction();

    Employee e1=new Employee();
    e1.setName("sonoo");

    Regular_Employee e2=new Regular_Employee();
    e2.setName("Vivek Kumar");
    e2.setSalary(50000);
    e2.setBonus(5);

    Contract_Employee e3=new Contract_Employee();
    e3.setName("Arjun Kumar");
    e3.setPay_per_hour(1000);
    e3.setContract_duration("15 hours");

    session.persist(e1);
    session.persist(e2);
    session.persist(e3);

    t.commit();
    session.close();
    System.out.println("success");
}
}
```

5. خروجی:

خروجی برنامه به صورت زیر خواهد بود:

EDIT	ID	TYPE	NAME	SALARY	BONUS	PAY_PER_HOUR	CONTRACT_DURATION
	1	emp	sonoo	-	-	-	-
	2	reg_emp	Vivek Kumar	50000	5	-	-
	3	con_emp	Arjun Kumar	-	-	1000	15 hours
							row(s) 1 - 3 of 3

آموزش مدل نگاشت Table Per Hierarchy به وسیلهAnnotation

در درس قبلی، ما به وسیله یک جدول تنها در یک فایلxml ، سلسله مراتب وراثت (inheritance hierarchy)را آدرس دهی و نگاشت (mapping) کردیم. در این درس، قصد داریم همین کار را مجددا به وسیله annotation انجام دهیم.

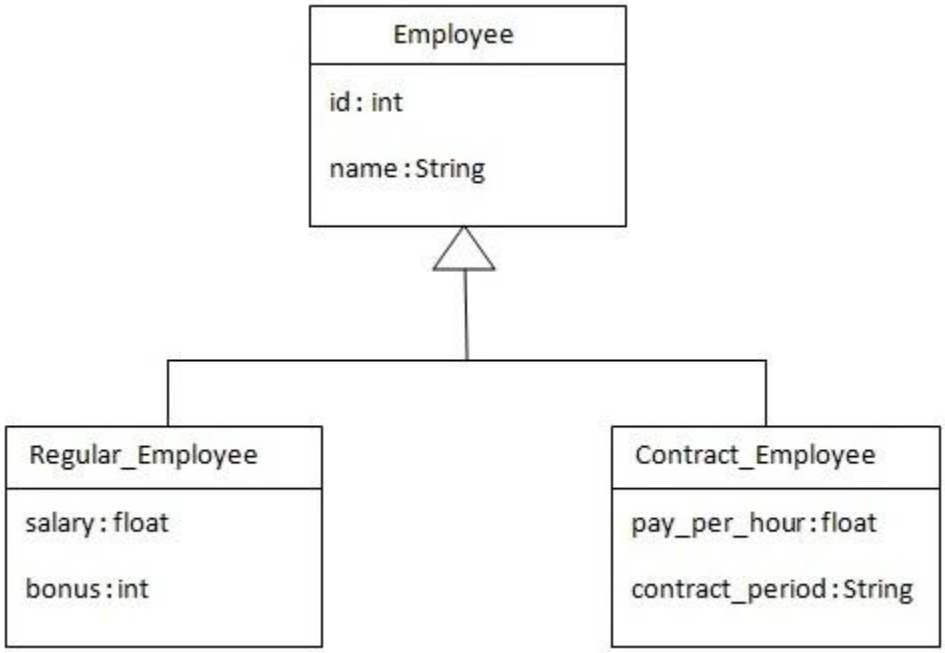
برای این منظور بایستی از موارد زیر در کد annotation خود استفاده کنید:

- @Inheritance
- @DiscriminatorColumn
- @DiscriminatorValue

برای انجام عمل نگاشتTable Per hierarchy ، فقط به یک جدول در پایگاه داده نیاز داریم.

همچنین این جدول دارای یک ستون اضافه به نام ستون تفکیک کننده (discriminator Column)می باشد که جهت تشخیص و متمایز کردن کلاس ها از هم به کار می رود.

نقشه سلسله مراتب وراثت در مدل مورد نظر ما به صورت زیر است:



3کلاس در سلسله مراتب (hierarchy) فوق وجود دارد. کلاس Employee که کلاس مادر و اصلی جهت کلاس های Regular_Employee و Contract_Employee است.

جدول ساختار وراثت کلاس های فوق به صورت زیر است:

Column Name	Data Type	Nullable	Default	Primary Key
ID	NUMBER(10,0)	No	-	1
TYPE	VARCHAR2(255)	No	-	-
NAME	VARCHAR2(255)	Yes	-	-
SALARY	FLOAT	Yes	-	-
BONUS	NUMBER(10,0)	Yes	-	-
PAY_PER_HOUR	FLOAT	Yes	-	-
CONTRACT_DURATION	VARCHAR2(255)	Yes	-	-
				1 - 7

مثال عملی نگاشت Table Per Hierarchy به وسیله Annotation

برای نگاشت مدل Table Per Hierarchy بایستی مراحل زیر را انجام دهید:

- ایجاد کلاس.Persistent Class
- ایجاد فایل تنظیمات و پیکربندی برنامه.Configuration file
- ایجاد کلاس لازم جهت نگهداری و تبادل داده ها.

1. ایجاد کلاس های Persistent Class لازم:

شما بایستی در ابتدا کلاس های اصلی و خام برنامه (Persistent Class) که سلسله مراتب وراثت را مشخص می کند، را ایجاد کنید. برای این منظور 3 کلاس زیر را تعریف می کنیم:

File: Employee.java

```
package com.javatpoint.mypackage;
import javax.persistence.*;

@Entity
@Table(name = "employee101")
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="type",discriminatorType=DiscriminatorType.STRING)
@DiscriminatorValue(value="employee")

public class Employee {
@Id
@GeneratedValue(strategy=GenerationType.AUTO)

@Column(name = "id")
private int id;

@Column(name = "name")
private String name;

//setters and getters
}
```

File: Regular_Employee.java

```
package com.javatpoint.mypackage;
import javax.persistence.*;

@Entity
@Table(name = "employee101")
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="type",discriminatorType=DiscriminatorType.STRING)
@DiscriminatorValue(value="employee")

public class Employee {
```

```
@Id
@GeneratedValue(strategy=GenerationType.AUTO)

@Column(name = "id")
private int id;

@Column(name = "name")
private String name;

//setters and getters
}
```

File: Contract_Employee.java

```
package com.javatpoint.mypackage;
import javax.persistence.*;

@Entity
@Table(name = "employee101")
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="type",discriminatorType=DiscriminatorType.STRING)
@DiscriminatorValue(value="employee")

public class Employee {
@Id
@GeneratedValue(strategy=GenerationType.AUTO)

@Column(name = "id")
private int id;

@Column(name = "name")
private String name;

//setters and getters
}
```

2. اضافه کردن Persistent Class در فایل تنظیمات برنامه:

فایل hibernate.cgf.xml را باز کرده و المنت های (mapping) زیر را بر هر یک از کلاس برنامه اضافه کنید.

کد فایل تنظیمات به صورت زیر بایستی تبدیل شود:

```
<mapping class="com.javatpoint.mypackage.Employee">
 <mapping class="com.javatpoint.mypackage.Contract_Employee">
 <mapping class="com.javatpoint.mypackage.Regular_Employee">
</mapping></mapping></mapping>
```

اکنون فایل تنظیمات به صورت زیر خواهد شد:

File: hibernate.cfg.xml

```
<!--?xml version='1.0' encoding='UTF-8'? -->


<!-- Generated by MyEclipse Hibernate Tools.  -->
<hibernate-configuration>

    <session-factory>
        <property name="hbm2ddl.auto">update</property>
        <property
name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
        <property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
        <property name="connection.username">system</property>
        <property name="connection.password">oracle</property>
        <property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>

        <mapping class="com.javatpoint.mypackage.Employee">
        <mapping class="com.javatpoint.mypackage.Contract_Employee">
        <mapping class="com.javatpoint.mypackage.Regular_Employee">
    </mapping></mapping></mapping></session-factory>

</hibernate-configuration>
```

نکته :

خاصیت hbm2ddl.auto برای تولید اتوماتیک جدول پایگاه داده تعیین شده است.

3. ایجاد کلاس لازم جهت نگهداری شی: Persistent Class Object

به وسیله کلاس StoreTest.java که کد آن در قسمت زیر مشخص شده است، می توانید شی employee را در پایگاه داده نگهداری کنید:

```
package com.javatpoint.mypackage;

import org.hibernate.*;
import org.hibernate.cfg.*;

public class StoreData {
public static void main(String[] args) {
    AnnotationConfiguration cfg=new AnnotationConfiguration();
    Session
session=cfg.configure("hibernate.cfg.xml").buildSessionFactory().openSession(
);
```

```
    Transaction t=session.beginTransaction();

    Employee e1=new Employee();
    e1.setName("sonoo");

    Regular_Employee e2=new Regular_Employee();
    e2.setName("Vivek Kumar");
    e2.setSalary(50000);
    e2.setBonus(5);

    Contract_Employee e3=new Contract_Employee();
    e3.setName("Arjun Kumar");
    e3.setPay_per_hour(1000);
    e3.setContract_duration("15 hours");

    session.persist(e1);
    session.persist(e2);
    session.persist(e3);

    t.commit();
    session.close();
    System.out.println("success");
}
}
```

4. خروجی برنامه به صورت زیر خواهد بود:

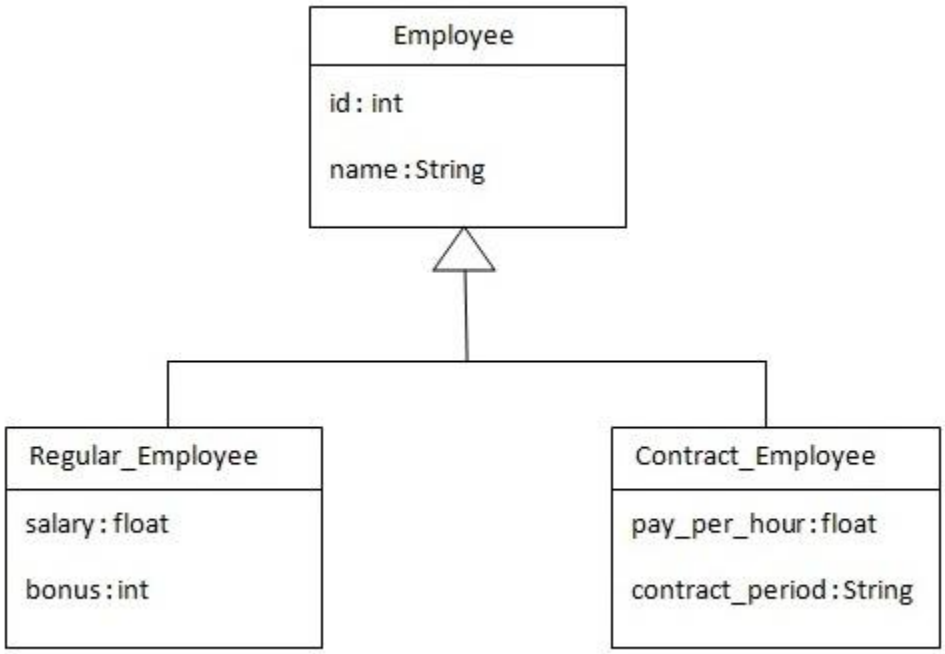
EDIT	ID	TYPE	NAME	SALARY	BONUS	PAY_PER_HOUR	CONTRACT_DURATION
	1	emp	sonoo	-	-	-	-
	2	reg_emp	Vivek Kumar	50000	5	-	-
	3	con_emp	Arjun Kumar	-	-	1000	15 hours
							row(s) 1 - 3 of 3

آموزش مدل نگاشت Table Per Concrete Class با استفاده از فایل xml

Cدر مدل نگاشت Table Per Concrete Class ، به ازای هر کلاس برنامه یک جدول در پایگاه داده ایجاد می شود که این جدول ها، هیچ نوع ارتباطی با هم ندارند. دو راه برای اتصال و ایجاد ارتباط بین جدول (Table) و مدل نگاشت Table Per Concrete Class وجود دارد که عبارتند از :

- استفاده از المنت Unio-Subclass.
- به وسیله ایجاد خود به خودی جدول برای هر کلاس.

ابتدا، بیایید نگاهی به کلاس های مثال درس و سلسله مراتب وراثت در آن ها بیاندازیم. دیاگرام زیر نشان دهنده کلاس ها و خواص آن هاست:



در کد فایل زیر، به وسیله المنت Union-subclass عمل نگاشت و آدرس دهی سلسله مراتب (hierarchy) را نشان داده ایم. ابتدا به مطالعه کد بپردازید، سپس به تشریح بخش های آن خواهیم پرداخت:

```
<!--?xml version='1.0' encoding='UTF-8'?-->

 <hibernate-mapping>
```

```
<class name="com.javatpoint.mypackage.Employee" table="emp122">
<id name="id">
<generator class="increment"></generator>
</id>

<property name="name"></property>

<union-subclass name="com.javatpoint.mypackage.Regular_Employee"
table="regemp122">
<property name="salary"></property>
<property name="bonus"></property>
</union-subclass>

<union-subclass name="com.javatpoint.mypackage.Contract_Employee"
table="contemp122">
<property name="pay_per_hour"></property>
<property name="contract_duration"></property>
</union-subclass>

</class>

</hibernate-mapping>
```

در مدل نگاشتTable Per Concrete Class ، 3جدول در پایگاه داده وجود خواهد داشت که هر کدام نماینده یکی از کلاس های برنامه است. زیرالمنت Union-Subclass، کلاس متعلق به جدول را تعیین می کند. این المنت، ستون های جدول Parent را به جدول اضافه کرده و همانند یک اتصال (Union) عمل می کند.

ساختار کلی هر یک از جدول های مثال به صورت زیر خواهد بود:

Table structure for Employee class

Column Name	Data Type	Nullable	Default	Primary Key
ID	NUMBER(10,0)	No	-	1
NAME	VARCHAR2(255)	Yes	-	-
				1 - 2

Table structure for Regular_Employee class

Column Name	Data Type	Nullable	Default	Primary Key
ID	NUMBER(10,0)	No	-	1
NAME	VARCHAR2(255)	Yes	-	-
SALARY	FLOAT	Yes	-	-
BONUS	NUMBER(10,0)	Yes	-	-
				1 - 4

Table structure for Contract_Employee class

Column Name	Data Type	Nullable	Default	Primary Key
ID	NUMBER(10,0)	No	-	1
NAME	VARCHAR2(255)	Yes	-	-
PAY_PER_HOUR	FLOAT	Yes	-	-
CONTRACT_DURATION	VARCHAR2(255)	Yes	-	-
				1 - 4

مثال عملی مدل نگاشت: Table Per Concrete Class

در این مثال عملی، 3 کلاس اصلی مثال را ایجاد کرده و سپس در فایل employee.bhm.xml، آدرس دهی و نگاشت (mapping) مورد نظر را انجام خواهیم داد.

1. ایجاد کلاس های: Persistent Class

در ابتدا شما بایستی کلاس های اصلی و خام را که تحت عنوان Persistent Class نامیده می شوند و وراثت را مشخص می سازند، ایجاد کنید. کد این 3 کلاس به ترتیب زیر است:

File: Employee.java

```
package com.javatpoint.mypackage;

public class Employee {
private int id;
private String name;

//getters and setters
}
```

File: Regular_Employee.java

```
package com.javatpoint.mypackage;

public class Regular_Employee extends Employee{
private float salary;
private int bonus;

//getters and setters
}
```

File: Contract_Employee.java

```
package com.javatpoint.mypackage;

public class Contract_Employee extends Employee{
    private float pay_per_hour;
    private String contract_duration;

//getters and setters
}
```

2. ایجاد فایل آدرس دهی (mapping) برای کلاس : Persistent Class نحوه آدرس دهی و نگاشت که در بخش قبل تشریح کردیم را به صورت زیر در فایل employee.bhm.xmlتعیین می کنیم:

فایل employee.bhm.xml :

```
<!--?xml version='1.0' encoding='UTF-8'?-->



 <hibernate-mapping>
 <class name="com.javatpoint.mypackage.Employee" table="emp122">
 <id name="id">
 <generator class="increment"></generator>
 </id>

 <property name="name"></property>

 <union-subclass name="com.javatpoint.mypackage.Regular_Employee"
table="regemp122">
 <property name="salary"></property>
 <property name="bonus"></property>
 </union-subclass>

 <union-subclass name="com.javatpoint.mypackage.Contract_Employee"
table="contemp122">
 <property name="pay_per_hour"></property>
 <property name="contract_duration"></property>
 </union-subclass>

 </class>
```

```
 </hibernate-mapping>
```

3. اضافه کردن آدرس دهی لازم جهت فایل hbm در فایل تنظیمات: Configuration file

فایل hibernate.xml را باز کرده و آدرس زیر را جهت mapping فایل hbm بر کد اضافه کنید:

```
<mapping resource="employee.hbm.xml">
</mapping>
```

اکنون کد فایل تنظیمات برنامه به صورت زیر خواهد شد:

```
<!--?xml version='1.0' encoding='UTF-8'?-->


<hibernate-configuration>

    <session-factory>
        <property name="hbm2ddl.auto">update</property>
        <property
name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
        <property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
        <property name="connection.username">system</property>
        <property name="connection.password">oracle</property>
        <property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
    <mapping resource="employee.hbm.xml">
    </mapping></session-factory>

</hibernate-configuration>
```

نکته :

خاصیت hbm2ddl.anto جهت تولید خودکار جدول ها در پایگاه داده در فایل تنظیمات تعیین شده است.

4. ایجاد کلاس لازم جهت نگهداری مشی employee در پایگاه داده:

در کلاس زیر به نامstoreData.java ، اقدام به نگهداری اشیای employee در پایگاه داده (database) می کنیم:

File: StoreData.java

```
package com.javatpoint.mypackage;

import org.hibernate.*;
import org.hibernate.cfg.*;
```

```
public class StoreData {
public static void main(String[] args) {
    Session session=new Configuration().configure("hibernate.cfg.xml")
                        .buildSessionFactory().openSession();

    Transaction t=session.beginTransaction();

    Employee e1=new Employee();
    e1.setName("sonoo");

    Regular_Employee e2=new Regular_Employee();
    e2.setName("Vivek Kumar");
    e2.setSalary(50000);
    e2.setBonus(5);

    Contract_Employee e3=new Contract_Employee();
    e3.setName("Arjun Kumar");
    e3.setPay_per_hour(1000);
    e3.setContract_duration("15 hours");

    session.persist(e1);
    session.persist(e2);
    session.persist(e3);

    t.commit();
    session.close();
    System.out.println("success");
}
}
```

آموزش مدل نگاشت Table Per Concrete Class با استفاده از Annotation

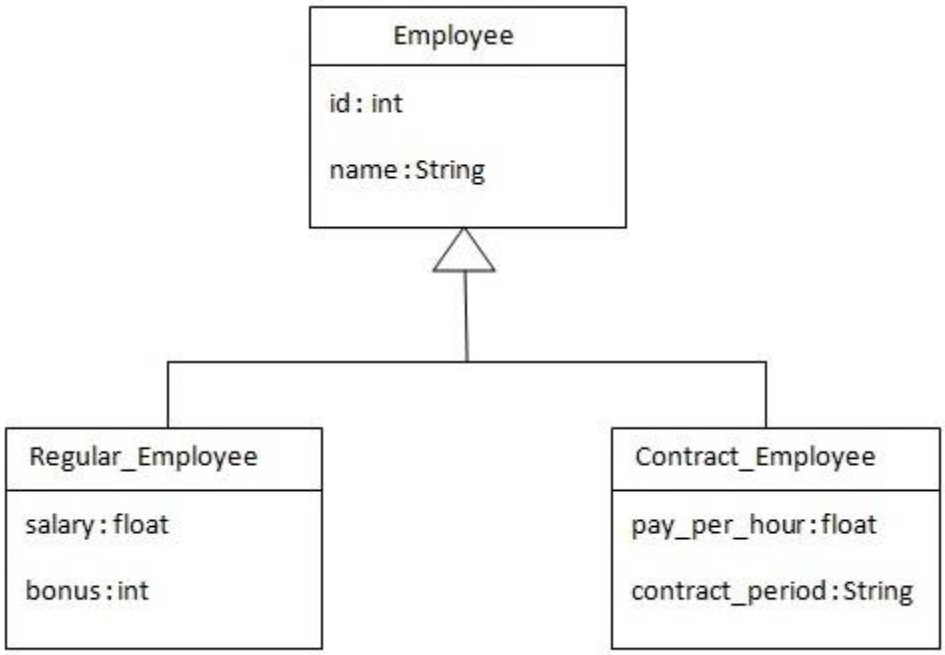
همان طور که در درس قبل اشاره کردیم، در مدل نگاشت Table Per Concrete Class به ازای هر کلاس، یک جدول در پایگاه داده ایجاد می‌شود. بنابراین مقادیر خالی (Nnallblc) در جدول ها نخواهیم داشت. تنها عیب استفاده از این روش، ایجاد ستون های تکراری در جدول های کلاس های زیرمجموعه است .

در این مدل ما بایستی از Annotation های Inheritance@ در کلاس اصلی و از AltributeOVerrides@ در کلاس های زیرمجموعه استفاده کنیم.

Inheritance@ مشخص می کند که ما داریم از مدل نگاشت Table Per Concrete Class در برنامه خود استفاده می کنیم. این Annotation فقط بایستی در کلاس Pareut تعیین شود.

از طرف دیگر، @AltributeOVerrides تعیین می کند که خواص کلاس Pareut در کلاس های زیرمجموعه، بازنویسی خواهند شد. در ساختار جدول ها، ستون های جدول کلاس Pareut در جدول کلاس زیرمجموعه، اضافه خواهند شد.

دیاگرام زیر نشان داده سلسله مراتب وراثت در کلاس های برنامه است:



ساختار جدول های هر یک از کلاس های مثال به صورت زیر خواهد بود:

Table structure for Employee class

Column Name	Data Type	Nullable	Default	Primary Key
ID	NUMBER(10,0)	No	-	1
NAME	VARCHAR2(255)	Yes	-	-
				1 - 2

Table structure for Regular_Employee class

Column Name	Data Type	Nullable	Default	Primary Key
ID	NUMBER(10,0)	No	-	1
NAME	VARCHAR2(255)	Yes	-	-
SALARY	FLOAT	Yes	-	-
BONUS	NUMBER(10,0)	Yes	-	-
				1 - 4

Table structure for Contract_Employee class

Column Name	Data Type	Nullable	Default	Primary Key
ID	NUMBER(10,0)	No	-	1
NAME	VARCHAR2(255)	Yes	-	-
PAY_PER_HOUR	FLOAT	Yes	-	-
CONTRACT_DURATION	VARCHAR2(255)	Yes	-	-
				1 - 4

مثال عملی مدل نگاشت: Table Per Concrete Class

در این مثال عملی، 3 کلاس اصلی مثال را ایجاد کرده و سپس در فایل employee.bhm.xml، آدرس دهی و نگاشت (mapping) مورد نظر را انجام خواهیم داد.

1. ایجاد کلاس های Persistent Class :

در ابتدا شما بایستی کلاس های اصلی و خام را که تحت عنوان Persistent Class نامیده می شوند و وراثت را مشخص می سازند، ایجاد کنید. کد این 3 کلاس به ترتیب زیر است:

File: Employee.java

```
package com.javatpoint.mypackage;

public class Employee {
private int id;
private String name;

//getters and setters
}
```

File: Regular_Employee.java

```
package com.javatpoint.mypackage;

public class Regular_Employee extends Employee{
private float salary;
private int bonus;

//getters and setters
}
```

File: Contract_Employee.java

```
package com.javatpoint.mypackage;

public class Contract_Employee extends Employee{
    private float pay_per_hour;
    private String contract_duration;

//getters and setters
}
```

2. ایجاد فایل آدرس دهی (mapping) برای کلاس : Persistent Class نحوه آدرس دهی و نگاشت که در بخش قبل تشریح کردیم را به صورت زیر در فایل employee.bhm.xml تعیین می کنیم:

فایل employee.hbm.xml :

```
<!--?xml version='1.0' encoding='UTF-8'?-->



  <hibernate-mapping>
  <class name="com.javatpoint.mypackage.Employee" table="emp122">
  <id name="id">
  <generator class="increment"></generator>
  </id>

  <property name="name"></property>

  <union-subclass name="com.javatpoint.mypackage.Regular Employee"
table="regemp122">
  <property name="salary"></property>
  <property name="bonus"></property>
  </union-subclass>

  <union-subclass name="com.javatpoint.mypackage.Contract_Employee"
table="contemp122">
  <property name="pay_per_hour"></property>
  <property name="contract_duration"></property>
  </union-subclass>
```

```
  </class>

  </hibernate-mapping>
```

3. اضافه کردن آدرس دهی لازم جهت فایل hbm در فایل تنظیمات: Configuration file

فایل hibernate.xml را باز کرده و آدرس زیر را جهت mapping فایل hbm بر کد اضافه کنید:

```
<mapping resource="employee.hbm.xml">
</mapping>
```

اکنون کد فایل تنظیمات برنامه به صورت زیر خواهد شد: hibernate.cfg.xml -

```
<!--?xml version='1.0' encoding='UTF-8'?-->


<hibernate-configuration>

    <session-factory>
        <property name="hbm2ddl.auto">update</property>
        <property
name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
        <property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
        <property name="connection.username">system</property>
        <property name="connection.password">oracle</property>
        <property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
    <mapping resource="employee.hbm.xml">
    </mapping></session-factory>

</hibernate-configuration>
```

نکته :

خاصیت hbm2ddl.anto جهت تولید خودکار جدول ها در پایگاه داده در فایل تنظیمات تعیین شده است.

4. ایجاد کلاس لازم جهت نگهداری مشی employee در پایگاه داده:

در کلاس زیر به نامstoreData.java ، اقدام به نگهداری اشیای employee در پایگاه داده (database) می کنیم:

File: StoreData.java

```
package com.javatpoint.mypackage;

import org.hibernate.*;
import org.hibernate.cfg.*;

public class StoreData {
public static void main(String[] args) {
    Session session=new Configuration().configure("hibernate.cfg.xml")
                        .buildSessionFactory().openSession();

    Transaction t=session.beginTransaction();

    Employee e1=new Employee();
    e1.setName("sonoo");

    Regular_Employee e2=new Regular_Employee();
    e2.setName("Vivek Kumar");
    e2.setSalary(50000);
    e2.setBonus(5);

    Contract_Employee e3=new Contract_Employee();
    e3.setName("Arjun Kumar");
    e3.setPay_per_hour(1000);
    e3.setContract_duration("15 hours");

    session.persist(e1);
    session.persist(e2);
    session.persist(e3);

    t.commit();
    session.close();
    System.out.println("success");
}
}
```

آموزش مدل نگاشت Table Per Subclass با استفاده از فایل xml

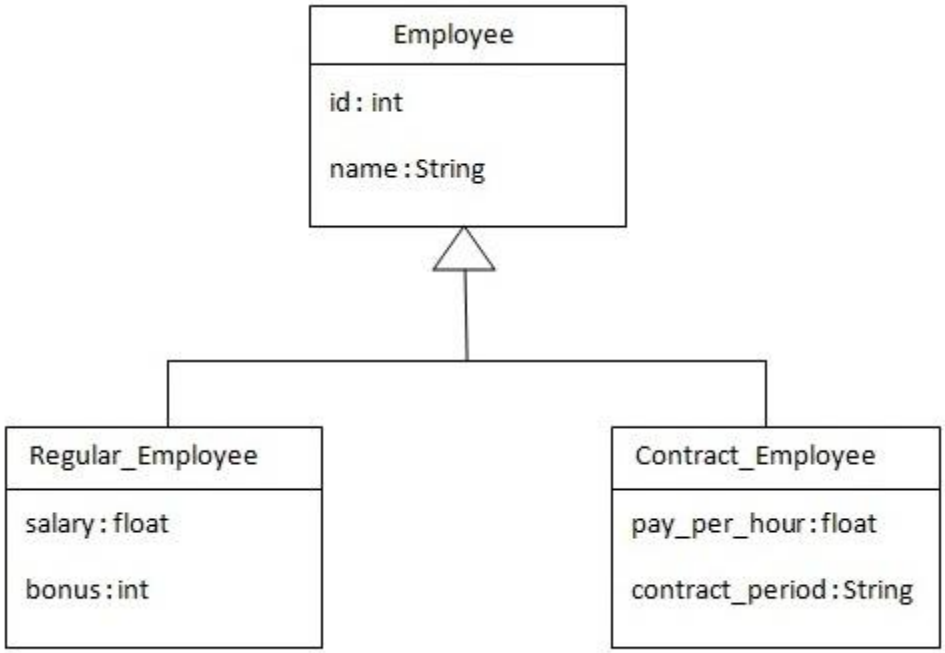
در مدل نگاشت Table Per Subclass ، جدول های آدرس دهی شده کلاس های زیرمجموعه از طریق کلیدهای اصلی (Primary key) و یا کلید خارجی (foreign key) به جدول کلاس اصلی Parent برنامه مرتبط هستند.

از المنت برای اتصال و آدرس دهی کلاس فرزند (child class) به کلاس مادر (Parent class) توسط Primary key یا foreign key استفاده می شود.

در مثال این درس، قصد داریم تا از خاصیت hb2dll.auto برای تولید خودکار جداول استفاده کنیم. بنابراین

نیازی نیست نگران ایجاد جدول ها در پایگاه داده باشیم.

ابتدا بیایید نگاهی به سلسله مراتب کلاس هایی که می خواهیم با هم نگاشت یا آدرس دهی کنیم، بپردازیم:



در کد فایل زیر، نحوه آدرس دهی (map) سلسله مراتب Hierarchy را با استفاده از المنت-joined subclassنشان داده ایم:

```
<!--?xml version='1.0' encoding='UTF-8'?-->



<hibernate-mapping>
<class name="com.javatpoint.mypackage.Employee" table="emp123">
<id name="id">
<generator class="increment"></generator>
</id>

<property name="name"></property>

<joined-subclass name="com.javatpoint.mypackage.Regular_Employee"
table="regemp123">
<key column="eid"></key>
<property name="salary"></property>
<property name="bonus"></property>
</joined-subclass>

<joined-subclass name="com.javatpoint.mypackage.Contract_Employee"
table="contemp123">
```

```
<key column="eid"></key>
<property name="pay_per_hour"></property>
<property name="contract_duration"></property>
</joined-subclass>

</class>
</hibernate-mapping>
```

در مدل نگاشتTable Per Subclass ، سه جدول در پایگاه داده وجود دارد که هر یک از آن ها نماینده یکی از کلاس های برنامه می باشد. همچنین زیر المنت joined-Subclass، کلاس زیرمجموعه یا Subclass را در کد مشخص می کند. ساختار هر یک از جدول های سلسله مراتب برنامه به صورت زیر است:

Table structure for Employee class

Column Name	Data Type	Nullable	Default	Primary Key
ID	NUMBER(10,0)	No	-	1
NAME	VARCHAR2(255)	Yes	-	-
				1 - 2

Table structure for Regular_Employee class

Column Name	Data Type	Nullable	Default	Primary Key
EID	NUMBER(10,0)	No	-	1
SALARY	FLOAT	Yes	-	-
BONUS	NUMBER(10,0)	Yes	-	-
				1 - 3

Table structure for Contract_Employee class

Column Name	Data Type	Nullable	Default	Primary Key
EID	NUMBER(10,0)	No	-	1
PAY_PER_HOUR	FLOAT	Yes	-	-
CONTRACT_DURATION	VARCHAR2(255)	Yes	-	-
				1 - 3

مثال عملی نگاشت با مدل Table Per SubClass Class

در مثال عملی این درس، ابتدا سه کلاس اصلی مثال را طراحی کرده و سپس آدرس دهی (mapping) آن ها را در فایل employee.hbm.xml تعیین می کنیم . برای این منظور مراحل زیر را انجام دهید:

1. ایجاد کلاس های اصلی Persistent Class

در مرحله اول، بایستی کلاس های اصلی و خام برنامه (Persistant Class) ها را که سلسله مراتب وراثت (hierarchy) را مشخص می کنند، تولید شود. فایل های زیر که هر کلاس را مشخص می کند:

File: Employee.java

```
package com.javatpoint.mypackage;

public class Employee {
private int id;
private String name;

//getters and setters
}
```

File: Regular_Employee.java

```
package com.javatpoint.mypackage;
public class Regular_Employee extends Employee{
private float salary;
private int bonus;

//getters and setters
}
```

File: Contract_Employee.java

```
package com.javatpoint.mypackage;

public class Contract_Employee extends Employee{
    private float pay_per_hour;
    private String contract_duration;

//getters and setters
}
```

2. ایجاد فایل نقشه دهی (mapping) برای کلاس Persistent

کد فایل employee.hbm.xml که سلسله مراتب وراثت را در برنامه مشخص می کند، به شرح زیر است:

```
<!--?xml version='1.0' encoding='UTF-8'?-->



<hibernate-mapping>
<class name="com.javatpoint.mypackage.Employee" table="emp123">
<id name="id">
<generator class="increment"></generator>
</id>

<property name="name"></property>

<joined-subclass name="com.javatpoint.mypackage.Regular_Employee"
table="regemp123">
<key column="eid"></key>
<property name="salary"></property>
<property name="bonus"></property>
</joined-subclass>

<joined-subclass name="com.javatpoint.mypackage.Contract_Employee"
table="contemp123">
<key column="eid"></key>
<property name="pay_per_hour"></property>
<property name="contract_duration"></property>
</joined-subclass>

</class>
</hibernate-mapping>
```

3. ایجاد فایل تنظیمات برنامه Configuration file

فایل تنظیمات برنامه hibernate.cgf.xml را باز کرده و المنت زیر را جهت آدرس دهی (mapping) به کد فایل اضافه کنید:

```
<mapping resource="employee.hbm.xml">              </mapping>
```

پس از اصلاح کد فایل بایستی به صورت زیر تبدیل شود:

```
<!--?xml version='1.0' encoding='UTF-8'?-->


<hibernate-configuration>
```

```
    <session-factory>
        <property name="hbm2ddl.auto">update</property>
        <property
name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
        <property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
        <property name="connection.username">system</property>
        <property name="connection.password">oracle</property>
        <property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
    <mapping resource="employee.hbm.xml">
    </mapping></session-factory>

</hibernate-configuration>
```

توجه :

خاصیت hbm2ddl.auto برای تولید اتوماتیک جدول ها در پایگاه داده تعین شده است.

4. ایجاد کلاس لازم جهت نگهداری شی Persistent Object

در کلاس StoreData.java که کد آن را در بخش زیر مشاهده می کنید، ساز و کار لازم جهت نگهداری شی Persistent Objectلحاظ شده است:

```
package com.javatpoint.mypackage;

import org.hibernate.*;
import org.hibernate.cfg.*;

public class StoreData {
public static void main(String[] args) {
    Session session=new Configuration().configure("hibernate.cfg.xml")
                        .buildSessionFactory().openSession();

    Transaction t=session.beginTransaction();

    Employee e1=new Employee();
    e1.setName("sonoo");

    Regular_Employee e2=new Regular_Employee();
    e2.setName("Vivek Kumar");
    e2.setSalary(50000);
    e2.setBonus(5);

    Contract_Employee e3=new Contract_Employee();
    e3.setName("Arjun Kumar");
    e3.setPay_per_hour(1000);
    e3.setContract_duration("15 hours");

    session.persist(e1);
```

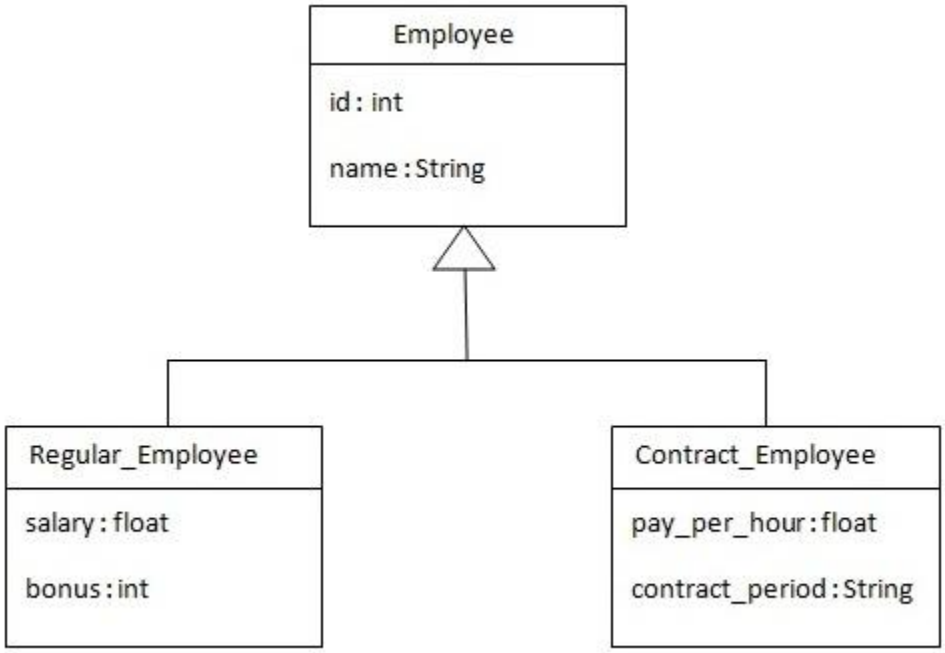
```
    session.persist(e2);
    session.persist(e3);

    t.commit();
    session.close();
    System.out.println("success");
}
}
```

آموزش مدل نگاشت Table Per Subclass با استفاده ازAnnotation

همان طور که در درس های قبل، اشاره کردیم در استراتژی نگاشتTable Per Subclass ، جدول های پایگاه داده به ازای هر کلاس اصلی (Persistent Class) ایجاد شده و از طریق کلیدهای اصلی (Primary key) و کلیدهای خارجی (Foreign key) به هم متصل هستند. بنابراین در این نوع رابطه، ستون های تکراری نخواهیم داشت.

در این مدل نیاز داریم تا annotation های Inheritance@ را در کلاس مادر (Parent) و PrimaryKeyJoinColumn@را در کلاس های زیرمجموعه (Subclass) تعریف کنیم.

ابتدا بیایید به سلسله مراتب کلاس هایی که می خواهیم آن ها را آدرس دهی یا map کنیم، نگاهی بیاندازیم :



ساختار هر یک از جدول به شرح زیر خواهد بود.

Table structure for Employee class

Column Name	Data Type	Nullable	Default	Primary Key
ID	NUMBER(10,0)	No	-	1
NAME	VARCHAR2(255)	Yes	-	-
				1 - 2

Table structure for Regular_Employee class

Column Name	Data Type	Nullable	Default	Primary Key
ID	NUMBER(10,0)	No	-	1
NAME	VARCHAR2(255)	Yes	-	-
SALARY	FLOAT	Yes	-	-
BONUS	NUMBER(10,0)	Yes	-	-
				1 - 4

Table structure for Contract_Employee class

Column Name	Data Type	Nullable	Default	Primary Key
ID	NUMBER(10,0)	No	-	1
NAME	VARCHAR2(255)	Yes	-	-
PAY_PER_HOUR	FLOAT	Yes	-	-
CONTRACT_DURATION	VARCHAR2(255)	Yes	-	-
				1 - 4

مثال عملی مدل نگاشت Table Per Subclass با استفاده ازAnnotation

در این مثال عملی، سه جدول اصلی برنامه را ایجاد کرده و سپس آدرس دهی (mapping)لازم برای آن ها را در فایل employee.xml انجام خواهیم داد.

1. ایجاد کلاس های اصلی Persistent Class

در مرحله اول، بایستی کلاس های اصلی برنامه که وراثت را تولید می کنند، ایجاد کنیم.

کد کلاسEmployee.java

```
package com.javatpoint.mypackage;
import javax.persistence.*;

@Entity
@Table(name = "employee103")
@Inheritance(strategy=InheritanceType.JOINED)

public class Employee {
@Id
@GeneratedValue(strategy=GenerationType.AUTO)

@Column(name = "id")
private int id;

@Column(name = "name")
private String name;

//setters and getters
}
```

کد کلاسRegular_Employee.java

```
package com.javatpoint.mypackage;

import javax.persistence.*;
```

```
@Entity
@Table(name="regularemployee103")
@PrimaryKeyJoinColumn(name="ID")
public class Regular_Employee extends Employee{

@Column(name="salary")
private float salary;

@Column(name="bonus")
private int bonus;

//setters and getters
}
```

کد کلاس Contract_Employee.java

```
package com.javatpoint.mypackage;

import javax.persistence.*;

@Entity
@Table(name="contractemployee103")
@PrimaryKeyJoinColumn(name="ID")
public class Contract_Employee extends Employee{

    @Column(name="pay_per_hour")
    private float pay_per_hour;

    @Column(name="contract_duration")
    private String contract_duration;

    //setters and getters
}
```

2. ایجاد فایل تنظیمات و پیکربندی اطلاعات برنامه: Congiguratio File

فایل hibernate.cgf.xml را باز کرده و المنت های زیر را جهت آدرس دهی هر یک از کلاس های برنامه به کد آن اضافه کنید:

```
<mapping class="com.javatpoint.mypackage.Employee">
<mapping class="com.javatpoint.mypackage.Contract_Employee">
<mapping class="com.javatpoint.mypackage.Regular_Employee">
            </mapping></mapping></mapping>
```

پس از این تغییر، کد فایل بایستی به صورت زیر شود:

```
<!--?xml version='1.0' encoding='UTF-8'?-->


<!-- Generated by MyEclipse Hibernate Tools.                    -->
<hibernate-configuration>

    <session-factory>
        <property name="hbm2ddl.auto">update</property>
        <property
name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
        <property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
        <property name="connection.username">system</property>
        <property name="connection.password">oracle</property>
<property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>

        <mapping class="com.javatpoint.mypackage.Employee">
        <mapping class="com.javatpoint.mypackage.Contract_Employee">
        <mapping class="com.javatpoint.mypackage.Regular_Employee">
    </mapping></mapping></mapping></session-factory>

</hibernate-configuration>
```

نکته :

خاصیت hbm2ddl.auto جهت تولید اتوماتیک جدول ها در پایگاه داده، به فایل تنظیمات برنامه اضافه شده است.

3. ایجاد کلاس لازم جهت نگهداری شی: Persistant object

در کلاس StoreData.java که کد آن را مشاهده می کنیم، ساز و کار لازم جهت نگهداری شی Persistant object تعریف شده است.

```
package com.javatpoint.mypackage;
import org.hibernate.*;
import org.hibernate.cfg.*;

public class StoreData {
public static void main(String[] args) {
    AnnotationConfiguration cfg=new AnnotationConfiguration();
    Session
session=cfg.configure("hibernate.cfg.xml").buildSessionFactory().openSession(
);

    Transaction t=session.beginTransaction();

    Employee e1=new Employee();
```

```
    e1.setName("sonoo");

    Regular_Employee e2=new Regular_Employee();
    e2.setName("Vivek Kumar");
    e2.setSalary(50000);
    e2.setBonus(5);

    Contract_Employee e3=new Contract_Employee();
    e3.setName("Arjun Kumar");
    e3.setPay_per_hour(1000);
    e3.setContract_duration("15 hours");

    session.persist(e1);
    session.persist(e2);
    session.persist(e3);

    t.commit();
    session.close();
    System.out.println("success");
}
}
```

آموزش آدرس دهی مجموعه ها (Collection) درHibernate

آموزش آدرس دهی مجموعه ها (Collection) در Hibernate شما می توانید عناصر مجموعه ای (Collection Elements) موجود در کلاس اصلی Persistent Class را در Hibernate آدرس دهی (mapping) کنید.

برای این منظور، بایستی نوع داده ای مجموعه را در کلاسPersistent Class ، از یکی از انواع زیر تعیین کنید:

- java.util.List
- java.util.Set
- java.util.SortedSet
- java.util.Map
- java.util.SortedMap
- vjava.util.Collection

- org.hibernate.usertype.UserCollectionType

کلاس اصلی و خام برنامه (Persistent Class) بایستی به صورت زیر برای عنصر مجموعه ای تعریف شود:

```
package com.javatpoint;

import java.util.List;

public class Question {
private int id;
private String qname;
private List answers;//List can be of any type

//getters and setters

}
```

آموزش روش آدرس دهی مجموعه در فایل mapping

عنصرهای زیادی در المنت هستند که از آن ها می توان برای آدرس دهی مجموعه استفاده کرد. این عناصر عبارتند از ، ، و .

در کد زیر نحوه آدرس دهی Persistent Class را به وسیله المنت های فوق نشان داده ایم:

```
<class name="com.javatpoint.Question" table="q100">
<id name="id">
<generator class="increment"></generator>
</id>
<property name="qname"></property>

<list name="answers" table="ans100">
<key column="qid"></key>
<index column="type"></index>
<element column="answer" type="string"></element>
</list>

</class>
```

در کد فوق 3 عنصر به کار رفته اند که عبارتند از:

- المنت برای تعیین کلید خارجی (Foreign key) در جدول فوق به کار رفته است.
- المنت برای تعیین نوع داده ای به کار رفته است List . و Map ، مجموعه های ایندکس شده هستند.
- المنت برای تعیین المنت اصلی مجموعه به کار می رود .

در کد زیر ، آدرس دهی mapping در صورتی که کلاس اشیای متنی (String objects) را نگهداری کند، تعیین شده است اما اگر مجموعه رفرنس موجودیت ها را نگهداری می کند (برای مثال اشیای یک کلاس دیگر)، بایستی المنت های و را در کد تعیین کنیم. در نهایت کد کلاس Persistent Class به صورت زیر خواهد شد:

```
package com.javatpoint;

import java.util.List;

public class Question {
private int id;
private String qname;
private List answers;//Here, List stores the objects of Answer class

//getters and setters

}

package com.javatpoint;
import java.util.List;
public class Answer {
private int id;
private String answer;
private String posterName;
//getters and setters
}
```

همچنین فایل آدرس دهی mapping نیز به صورت زیر می باشد:

```
<class name="com.javatpoint.Question" table="q100">
<id name="id">
<generator class="increment"></generator>
</id>
<property name="qname"></property>

<list name="answers">
<key column="qid"></key>
<index column="type"></index>
<one-to-many class="com.javatpoint.Answer">
</one-to-many></list>

</class>
```

در کد فوق، Listبا روش ارتباط one-to-many (یک به چند) آدرس دهی شده است. در این روش، ممکن است جواب های مختلفی برای یک سوال وجود داشته باشد.

فهمیدن عنصر key

عنصر key برای تعیین کلید خارجی Foreign key در جدول های متصل به هم بر مبنای شناسه اصلی به کار می رود. عنصر Foreign key به صورت پیش فرض nullable است. بنابراین برای کلیدهای خارجی غیر nullable، بایستی خاصیت not-null را به صورت زیر تعیین کنیم:

```
<key column="qid" not-null="true"></key>
```

خواص مهم عنصر key مقادیر column ، an-delete، Unique، Update، not-null و Property-ref هستند که به صورت نمونه می توان آن ها را همانند کد زیر تعریف نمود:

```
<key column="columnname" on-delete="noaction|cascade" not-null="true|false"
property-ref="propertyName" update="true|false" unique="true|false">
            </key>
```

مجموعه های ایندکس شده یا indexed collections

به طور کلی، مجموعه ها به دو دسته تقسیم می شوند:

- مجموعه های ایندکس شده یا indexed Collections
- مجموعه های غیر ایندکسی یا non-indexed Collections

برای مثال مجموعه های List و Map از نوع indexed و مجموعه های bag و set از نوع non-indexed هستند.

در اینجا، مجموعه های indexed به معنای آن است که در مجموعه بایستی یک المنت اضافه به نام نیز تعیین شود.

المنت های مجموعه ای یا Collection Elements

المنت های مجموعه ای می توانند شامل مقادیر (Value) و یا رفرس موجودیت ها (اشیای یک کلاس دیگر) شوند. به طور کلی، ما 4 نوع مجموعه داریم:

- Element

- Component-element
- One-to-many
- Many-to-many

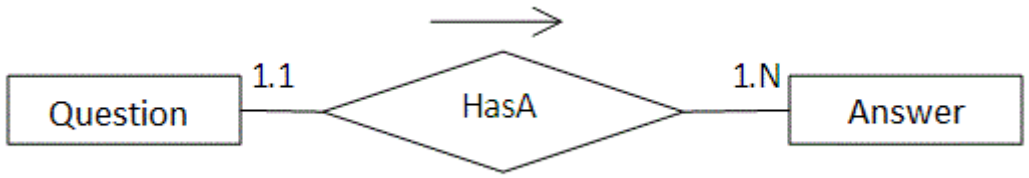
المنت های Component-element و element برای مقادیر معمولی مثل String یا int به کار می روند اما المنت های one-to-many و many-to-many برای آدرس دهی و فرض موجودیت ها (entity refrence) به کار می روند.

در درس های بعدی، به بررسی کار با المنت هایList ، Bag وSet خواهیم پرداخت.

آموزش آدرس دهی المنت List در آدرس دهی مجموعه ها:

اگر کلاس خام و اصلی برنامه(Persistant Class) ، حاوی شی List object باشد، به سادگی می توانید شی Listرا به وسیله المنت کلاس در فایل mapping و یا annotation آدرس دهی کنید.

در این درس، ما از سناریوی 1 یا Forum استفاده می کنیم که در آن هر سوال می تواند دارای چندین جواب باشد. شکل زیر بیان کننده مفهوم مسئله است:



به وسیله کد زیر، می توانیم شی List را در فایل mapping تعریف و تنظیم کنیم:

```
<class name="com.javatpoint.Question" table="q100">
        ...
           <list name="answers" table="ans100">
           <key column="qid"></key>
           <index column="type"></index>
           <element column="answer" type="string"></element>
           </list>

        ...
```

```
</class>
```

مثال عملی آموزش آدرس دهی المنت List در مجموعه ها

در این مثال عملی، قصد داریم تا نحوه آدرس دهی مجموعه ها را با المنت listبه طور کامل آموزش دهیم. در این مثال، المنت List مجموعه ای از مقادیر متنی String Values را نگهداری می کند نه رفرنس ها به موجودیت ها. به همین دلیل از روش element به جای one-to-many برای عنصر list استفاده می کنیم.

1. ایجاد کلاس اصلی و خام برنامه Persistent Class

کلاس اصلیPersistent Class ، کلید خواص Class از جمله خاصیت List را تعیین می کند. به صورت زیر:

```
package com.javatpoint;

import java.util.List;

public class Question {
private int id;
private String qname;
private List answers;

//getters and setters

}
```

2. ایجاد فایل آدرس دهی mapping file برای کلاس Persistent

در کد زیر، محتویات فایل question.hbm.xml را برای تعریف شی List قرار داده ایم:

```
<!--?xml version='1.0' encoding='UTF-8'?-->


<hibernate-mapping>
 <class name="com.javatpoint.Question" table="q100">
   <id name="id">
     <generator class="increment"></generator>
   </id>
   <property name="qname"></property>

   <list name="answers" table="ans100">
     <key column="qid"></key>
     <index column="type"></index>
```

```
     <element column="answer" type="string"></element>
   </list>

 </class>

</hibernate-mapping>
```

3. ایجاد فایل تنظیمات برنامه: Configuration file

فایل زیر که فایل تنظیمات برنامه است، شامل اطلاعاتی راجع به پایگاه داده و فایل آدرس دهی mapping file است.

```
<!--?xml version='1.0' encoding='UTF-8'?-->


<!-- Generated by MyEclipse Hibernate Tools.                    -->
<hibernate-configuration>

    <session-factory>
        <property name="hbm2ddl.auto">update</property>
        <property
name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
        <property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
        <property name="connection.username">system</property>
        <property name="connection.password">oracle</property>
        <property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
    <mapping resource="question.hbm.xml">
    </mapping></session-factory>

</hibernate-configuration>
```

4. ایجاد کلاس لازم جهت نگهداری اطلاعات:

در کلاس زیر، اطلاعات مربوط به کلاس question را نگهداری می کنیم:

```
package com.javatpoint;

import java.util.ArrayList;

import org.hibernate.*;
import org.hibernate.cfg.*;

public class StoreData {
 public static void main(String[] args) {
    Session session=new Configuration().configure("hibernate.cfg.xml")
                        .buildSessionFactory().openSession();
    Transaction t=session.beginTransaction();
```

```
    ArrayList<string> list1=new ArrayList<string>();
    list1.add("java is a programming language");
    list1.add("java is a platform");

    ArrayList<string> list2=new ArrayList<string>();
    list2.add("Servlet is an Interface");
    list2.add("Servlet is an API");

    Question question1=new Question();
    question1.setQname("What is Java?");
    question1.setAnswers(list1);

    Question question2=new Question();
    question2.setQname("What is Servlet?");
    question2.setAnswers(list2);

    session.persist(question1);
    session.persist(question2);

    t.commit();
    session.close();
    System.out.println("success");
  }
}
            </string></string></string></string>
```

5. آموزش نحوه دریافت اطلاعات از شی List

در کد زیر، ما از زبان HQL برای خواندن کلید رکوردهای کلاس Question از جمله answers استفاده کرده ایم. در این روش، برنامه اطلاعات رکوردها را از دو جدول که دارای عملکردی مستقل هستند، می خواند:

```
package com.javatpoint;

import java.util.*;

import org.hibernate.*;
import org.hibernate.cfg.*;

public class FetchData {
public static void main(String[] args) {

    Session session=new Configuration().configure("hibernate.cfg.xml")
                        .buildSessionFactory().openSession();

    Query query=session.createQuery("from Question");
    List<question> list=query.list();

    Iterator<question> itr=list.iterator();
    while(itr.hasNext()){
```

```
        Question q=itr.next();
        System.out.println("Question Name: "+q.getQname());

        //printing answers
        List<string> list2=q.getAnswers();
        Iterator<string> itr2=list2.iterator();
        while(itr2.hasNext()){
            System.out.println(itr2.next());
        }

    }
    session.close();
    System.out.println("success");

}
}
            </string></string></question></question>
```

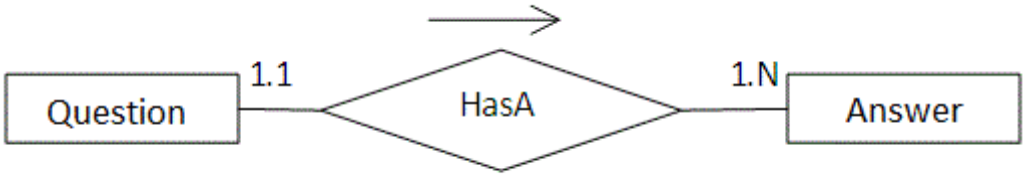
خروجی مثال هم به صورت زیر خواهد بود :

```
Question Name: What is Java?
java is a programming language by: Ravi Malik
java is a platform by: Sudhir Kumar
Question Name: What is Servlet?
Servlet is an Interface by: Jai Kumar
Servlet is an API by: Arun
success
```

آموزش آدرس دهی به روش One to Many در Hibernate با استفاده از مثال List در فایل xml

اگر کلاس Persistant Class شما دارای اشیای لیست آدرس دهی (object list) باشد که حاوی رفرنس های entity هستند، بایستی از روش one-to-many برای آدرس دهی (map) المنت های لیست دار استفاده کنیم.

در این مثال، ما از سناریو انجمن استفاده می کنیم که هر سوال می تواند چندین جواب داشته باشد. دیاگرام این سناریو در مثال زیر نشان داده شده است:



در مواردی همانند مثال فوق، برای هر سوال ممکن است چندین جواب وجود داشته باشد و هر جواب هم اطلاعات مختص به خود را دارد. برای همین دلیل ما از List جهت کلاس persistant Class استفاده کردیم. (شامل رفرنس های لازم به کلاس Answer) تا بتوانند جانشین مجموعه ای از جواب ها باشد. کد زیر، کد کلاس Persistant Classرا نشان می دهد که حاوی اشیای لیست دار یا list objects بوده که خود شامل اشیای کلاس Answer می باشد:

```
package com.javatpoint;

import java.util.List;

public class Question {
private int id;
private String qname;
private List<answer> answers;
//getters and setters

}
            </answer>
```

از طرف دیگر، کلاس Answer که کد آن به صورت زیر است، خود حاوی اطلاعاتی مثلid ، answername، postedBy و ... می باشد:

```
package com.javatpoint;

public class Answer {
private int id;
private String answername;
private String postedBy;
//getters and setters

}
}
```

همچنین کلاس Question دارای اشیای لیست دار list objects می باشد که حاوی رفرنس های entity هستند) برای مثال رفرنس به شی کلاس .(Answer در این چنین مواردی، بایستی از روش one-to-many برای آدرس دهی (map) این object استفاده کنیم. در کد زیر نحوه آدرس دهی نشان داده شده است:

```
<list name="answers" cascade="all">
          <key column="qid"></key>
          <index column="type"></index>
          <one-to-many class="com.javatpoint.Answer">
</one-to-many></list>
```

مثال آدرس دهی (mapping) با روش one to many در Hibernate با استفاده از List

در این مثال، قصد داریم تا روش کامل آدرس دهی (mapping) یک list که حاوی رفرنس های entity می باشد را آموزش دهیم. برای این منظور مراحل زیر را انجام دهید:

1. ایجاد کلاس Persistant Class

کلاس Persistant Class خواص (Properties) لازم جهت کلاس از جمله List را به صورت زیر تعیین می کند:

کد فایل: Question.java

```
package com.javatpoint;

import java.util.List;

public class Question {
private int id;
private String qname;
private List<answer> answers;

//getters and setters

}

            </answer>
```

کد فایلAnswer.java

```
package com.javatpoint;
```

```
public class Answer {
private int id;
private String answername;
private String postedBy;
//getters and setters

}
}
```

2. ایجاد فایل آدرس دهی (mapping) لازم جهت کلاس Persistant

در این مرحله، ما فایل question.hbm.xml را برای تعیین list ایجاد کرده ایم. به صورت زیر:

```
<!--?xml version='1.0' encoding='UTF-8'?-->


          <hibernate-mapping>
          <class name="com.javatpoint.Question" table="q501">
          <id name="id">
          <generator class="increment"></generator>
          </id>
          <property name="qname"></property>

          <list name="answers" cascade="all">
          <key column="qid"></key>
          <index column="type"></index>
          <one-to-many class="com.javatpoint.Answer">
          </one-to-many></list>

          </class>

          <class name="com.javatpoint.Answer" table="ans501">
          <id name="id">
          <generator class="increment"></generator>
          </id>
          <property name="answername"></property>
          <property name="postedBy"></property>
          </class>

</hibernate-mapping>
```

3. ایجاد فایل پیکربندی اطلاعات Configuration File

فایل زیر حاوی اطلاعات لازم جهت کار با database و فایل آدرس دهی mapping file است:

```
<!--?xml version='1.0' encoding='UTF-8'?-->


<!-- Generated by MyEclipse Hibernate Tools.                    -->
```

```
<hibernate-configuration>

    <session-factory>
        <property name="hbm2ddl.auto">update</property>
        <property
name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
        <property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
        <property name="connection.username">system</property>
        <property name="connection.password">oracle</property>
        <property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
    <mapping resource="question.hbm.xml">
    </mapping></session-factory>

</hibernate-configuration>
```

4. ایجاد کلاس لازم جت نگهداری اطلاعات:

در کلاس زیر نیز اقدام به نگهداری اطلاعات کلاس question class خواهیم پرداخت:

```
package com.javatpoint;
import java.util.ArrayList;
import org.hibernate.*;
import org.hibernate.cfg.*;

public class StoreData {
public static void main(String[] args) {
    Session session=new Configuration().configure("hibernate.cfg.xml")
                            .buildSessionFactory().openSession();
    Transaction t=session.beginTransaction();

    Answer ans1=new Answer();
    ans1.setAnswername("java is a programming language");
    ans1.setPostedBy("Ravi Malik");

    Answer ans2=new Answer();
    ans2.setAnswername("java is a platform");
    ans2.setPostedBy("Sudhir Kumar");

    Answer ans3=new Answer();
    ans3.setAnswername("Servlet is an Interface");
    ans3.setPostedBy("Jai Kumar");

    Answer ans4=new Answer();
    ans4.setAnswername("Servlet is an API");
    ans4.setPostedBy("Arun");

    ArrayList<answer> list1=new ArrayList<answer>();
    list1.add(ans1);
    list1.add(ans2);

    ArrayList<answer> list2=new ArrayList<answer>();
    list2.add(ans3);
```

```
    list2.add(ans4);

    Question question1=new Question();
    question1.setQname("What is Java?");
    question1.setAnswers(list1);

    Question question2=new Question();
    question2.setQname("What is Servlet?");
    question2.setAnswers(list2);

    session.persist(question1);
    session.persist(question2);

    t.commit();
    session.close();
    System.out.println("success");
}
}
            </answer></answer></answer></answer>
```

5. خروجی

ID	ANSWERNAME	POSTEDBY	QID	TYPE
1	java is a programming language	Ravi Malik	1	0
2	java is a platform	Sudhir Kumar	1	1
3	Servlet is an Interface	Jai Kumar	2	0
4	Servlet is an API	Arun	2	1

آموزش نحوه دریافت (fetch) اطلاعات لیست List

در این بخش، ما از زبان HQL برای دریافت اطلاعات کلیه رکوردهای کلاس Questionکه حاوی answers نیز هستند، استفاده می کنیم. در این مثال، کد ما اطلاعات را از دو جدول کد به صورت functional depandant به هم متصل هستند، استخراج می کند.

در کلاس FetchData.java ، شی کلاس answer class را به صورت مستقیم در خروجی چاپ می کنیم. اما ما متد t.String() در کلاس Answer Class را که خواص answernameو Poster name را بر می گرداند را بازنویسی کرده ایم. بنابراین، متد t.String() مقادیر answername و poster name را به جایid های رفرنس ها در خروجی چاپ می کند.

فایل FetchData.java

```
package com.javatpoint;
import java.util.*;
import org.hibernate.*;
import org.hibernate.cfg.*;

public class FetchData {
public static void main(String[] args) {

    Session session=new Configuration().configure("hibernate.cfg.xml")
                        .buildSessionFactory().openSession();

    Query query=session.createQuery("from Question");
    List<question> list=query.list();

    Iterator<question> itr=list.iterator();
    while(itr.hasNext()){
        Question q=itr.next();
        System.out.println("Question Name: "+q.getQname());

        //printing answers
        List<answer> list2=q.getAnswers();
        Iterator<answer> itr2=list2.iterator();
        while(itr2.hasNext()){
            System.out.println(itr2.next());
        }

    }
    session.close();
    System.out.println("success");
}
}
            </answer></answer></question></question>
```

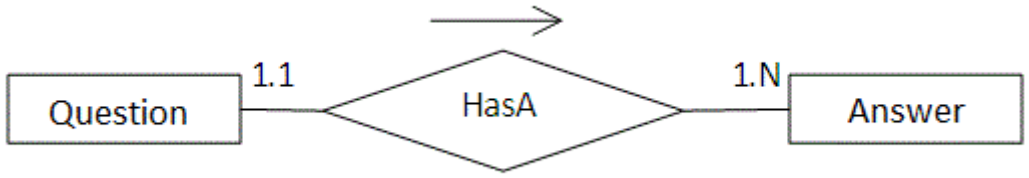
خروجی مثال هم به صورت زیر است:

```
Question Name: What is Java?
java is a programming language by: Ravi Malik
java is a platform by: Sudhir Kumar
Question Name: What is Servlet?
Servlet is an Interface by: Jai Kumar
Servlet is an API by: Arun
success
```

آموزش Mapping Bag در آدرس دهی مجموعه های Hibernate

اگر کلاس Persistent Class شما شامل اشیای لیست دار List object است، می توانید List را به وسیله المنت list یا bag در فایل آدرس دهی mapping file آدرس دهی یا map کنید. المنت bag کاملا مشابه المنت List می باشد با این تفاوت که نیاز به المنت index ندارد.

در این درس، همانند درس های قبل از سناریوی برنامه نویسی انجمن برای تشریح مثال خود استفاده می کنیم که در آن هر سوال می تواند چندین جواب داشته باشد. شکل زیر، نحوه عملکرد یک انجمن را نشان می دهد:



در کد زیر به آموزش نحوه پیاده سازی المنت bag در فایل آدرس دهی mapping file پرداخته ایم:

```
<class name="com.javatpoint.Question" table="q100">
        ...
           <bag name="answers" table="ans100">
           <key column="qid"></key>
           <element column="answer" type="string"></element>
           </bag>

        ...
</class>
```

مثال عملی Mapping bag در آدرس دهی مجموعه های Hibernate

در این مثال، قصد داریم تا به طور کامل نحوه آدرس دهی مجموعه های Hibernateبه وسیله bag را آموزش دهیم. در این مثالbag ، این عنصر مقایسه (Values)را نگهداری می کند تا رفرنس های entity را، به همین دلیل، ما از المنت element به جای one-to-many در کد خود استفاده می کنیم.

اگر مثال mapping list را در درس های قبل مشاهده کرده باشید، این مثال نیز کاملا همانند آن است به جز این که در mapping file به جای list از bag استفاده می کنیم.

برای اجرای مثال، مراحل زیر را به ترتیب انجام دهید:

1. ایجاد کلاس Persistent Class

کلاس Persistent Class کلیه خواص Properties لازم جهت class را از جمله List تعیین می کند، به صورت زیر:

```
package com.javatpoint;

import java.util.List;

public class Question {
private int id;
private String qname;
private List<string> answers;

//getters and setters

}
            </string>
```

2. ایجاد فایل آدرس دهی mapping file برای کلاس Persistent

در این بخش، فایل question .hbm.xml را برای تعیین list ایجاد کرده ایم:

```
<!--?xml version='1.0' encoding='UTF-8'?-->


<hibernate-mapping>
 <class name="com.javatpoint.Question" table="q101">
   <id name="id">
     <generator class="increment"></generator>
   </id>
   <property name="qname"></property>

   <bag name="answers" table="ans101">
     <key column="qid"></key>
     <element column="answer" type="string"></element>
   </bag>

 </class>

</hibernate-mapping>
```

3. ایجاد فایل پیکربندی اطلاعات Configuration file

فایل زیر، حاوی اطلاعات لازم درباره پایگاه داده و فایل آدرس دهی برنامه است:

```
<!--?xml version='1.0' encoding='UTF-8'?-->
```

```
<!-- Generated by MyEclipse Hibernate Tools.                    -->
<hibernate-configuration>

    <session-factory>
        <property name="hbm2ddl.auto">update</property>
        <property
name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
        <property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
        <property name="connection.username">system</property>
        <property name="connection.password">oracle</property>
        <property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
    <mapping resource="question.hbm.xml">
    </mapping></session-factory>

</hibernate-configuration>
```

4. ایجاد کلاس لازم جهت نگهداری اطلاعات:

در کلاس زیر نیز اقدام به نگهداری اطلاعات کلاس question class پرداخته ایم:

```
package com.javatpoint;

import java.util.ArrayList;

import org.hibernate.*;
import org.hibernate.cfg.*;

public class StoreData {
 public static void main(String[] args) {
    Session session=new Configuration().configure("hibernate.cfg.xml")
                        .buildSessionFactory().openSession();
    Transaction t=session.beginTransaction();

    ArrayList<string> list1=new ArrayList<string>();
    list1.add("java is a programming language");
    list1.add("java is a platform");

    ArrayList<string> list2=new ArrayList<string>();
    list2.add("Servlet is an Interface");
    list2.add("Servlet is an API");

    Question question1=new Question();
    question1.setQname("What is Java?");
    question1.setAnswers(list1);

    Question question2=new Question();
    question2.setQname("What is Servlet?");
    question2.setAnswers(list2);

    session.persist(question1);
    session.persist(question2);
```

```
    t.commit();
    session.close();
    System.out.println("success");
  }
}
                    </string></string></string></string>
```

آموزش نحوه استخراج (fetch) اطلاعات:

در این بخش، ما از زبان HQL برای استخراج و دریافت (fetch) اطلاعات کلیه رکوردهای کلاس Question از جمله answers پرداخته ایم. در چنین موردی، برنامه اطلاعات را از دو جدول مختلف کد به صورت functional dependent هستند، دریافت می کند.

```
package com.javatpoint;

import java.util.*;

import org.hibernate.*;
import org.hibernate.cfg.*;

public class FetchData {
public static void main(String[] args) {

    Session session=new Configuration().configure("hibernate.cfg.xml")
                        .buildSessionFactory().openSession();

    Query query=session.createQuery("from Question");
    List<question> list=query.list();

    Iterator<question> itr=list.iterator();
    while(itr.hasNext()){
        Question q=itr.next();
        System.out.println("Question Name: "+q.getQname());

        //printing answers
        List<string> list2=q.getAnswers();
        Iterator<string> itr2=list2.iterator();
        while(itr2.hasNext()){
            System.out.println(itr2.next());
        }

    }
    session.close();
    System.out.println("success");

}
}
            </string></string></question></question>
```

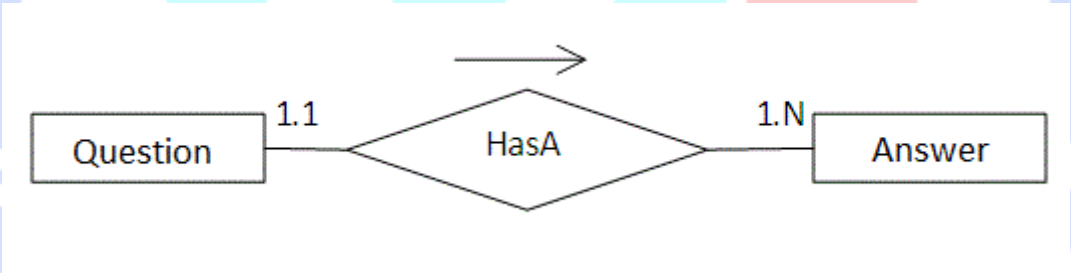
آموزش آدرس دهی mapping One to Many در Hibernate با استفاده از List

اگر کلاس Persistent Class شما شامل اشیای لیست دار list object ای باشد که خود حاوی رفرنس های entityاست، بایستی از روش آدرس دهی One to Many mapping برای آدرس دهی یا map کردن المنت های list استفاده کنیم. ما می توانیم این شی لیست (list object) را با استفاده از المنت list یا bag آدرس دهی کنیم.

نکته :

توجه داشته باشید که المنت bag ایندکس شده نیست(no indexed-based) ، در حالی که المنت list ایندکس شده است.

در این درس هم مانند درس های از قبل، از سناریوی انجمن برای تشریح مثال خود استفاده می کنیم که در آن هر سوال می تواند چندین جواب داشته باشد. شکل زیر دیاگرام کارکرد این مثال را نشان می دهد:



بیایید در کد زیر نگاهی به کلاس Persistent Class که حاوی اشیای لیست دار (list objects) بیاندازیم. در چنین موردی، می تواند برای هر سوال questionچندین جواب answer وجود داشته باشد و هر answer هم می تواند شامل اطلاعات خود باشد. برای همین دلیل است که ما از hist element های حاوس اشیای Answersبرای جایگزینی مجموعه ای از جواب ها استفاده می کنیم.

```
package com.javatpoint;
import java.util.List;
public class Question {
private int id;
private String qname;
private List<answer> answers;
//getters and setters
}
            </answer>
```

کلاس Answer Class که کد آن در بخش زیر نشان داده شده است حاوی اطلاعات مختص به خود از جمله id ، answername، PostedBy و ... می باشد:

```
package com.javatpoint;
public class Answer {
private int id;
private String answername;
private String postedBy;
//getters and setters
}
}
```

کلاس Question Class نیز حاوی اشیای لیست دار list object است که دربرگیرنده رفرنس های entity می باشند، برای مثال .Answer Class objects در چنین مواردی، ما بایستی از روش آدرس دهی one-to-many برای آدرس دهی یا map این شی استفاده کنیم. در کد زیر نحوه آدرس این object آموزش داده شده است:

```
<bag name="answers" cascade="all">
          <key column="qid"></key>
          <one-to-many class="com.javatpoint.Answer">

            </one-to-many></bag>
```

مثال عملی روش آدرس دهی one to many در مجموعه های Hibernate

در این مثال عملی، روش آدرس دهی list هایی که حاوی رفرنس های entity هستند را به طور کامل آموزش خواهیم داد. برای این منظور مراحل زیر را انجام دهید:

1. کلاس Persistent Class خواص یا Properties لازم جهت کلاس از جمله List را به صورت زیر تعیین می کند:

فایل Question.java

```
package com.javatpoint;

import java.util.List;

public class Question {
private int id;
private String qname;
```

```
private List answers;

//getters and setters

}
```

فایلAnswer.java

```
package com.javatpoint;

public class Answer {
private int id;
private String answername;
private String postedBy;
//getters and setters

}
}
```

2. ایجاد فایل آدرس دهی لازم (mapping file) برای کلاس Persistent

در این بخش، فایل question.hbm.xml را برای تعیین list به صورت زیر ایجاد می کنیم:

```
<!--?xml version='1.0' encoding='UTF-8'?-->


          <hibernate-mapping>
          <class name="com.javatpoint.Question" table="q501">
          <id name="id">
          <generator class="increment"></generator>
          </id>
          <property name="qname"></property>

          <bag name="answers" cascade="all">
          <index column="type"></index>
          <one-to-many class="com.javatpoint.Answer">
          </one-to-many></bag>

          </class>

          <class name="com.javatpoint.Answer" table="ans501">
          <id name="id">
          <generator class="increment"></generator>
          </id>
          <property name="answername"></property>
          <property name="postedBy"></property>
          </class>

</hibernate-mapping>
```

3. ایجاد فایل پیکربندی اطلاعات Configuration file

فایل زیر کد فایل Cinfiguration برنامه است، حاوی اطلاعات لازم درباره پایگاه داده و فایل آدرس دهی mapping file می باشد:

```
<!--?xml version='1.0' encoding='UTF-8'?-->


<!-- Generated by MyEclipse Hibernate Tools.                    -->
<hibernate-configuration>

    <session-factory>
        <property name="hbm2ddl.auto">update</property>
        <property
name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
        <property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
        <property name="connection.username">system</property>
        <property name="connection.password">oracle</property>
        <property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
    <mapping resource="question.hbm.xml">
    </mapping></session-factory>

</hibernate-configuration>
```

4. ایجاد کلاس لازم برای نگهداری اطلاعات برنامه:

در کلاس زیر به نام StoreData.java کد لازم جهت نگهداری اطلاعات برنامه را تعیین کرده ایم:

```
package com.javatpoint;

import java.util.ArrayList;

import org.hibernate.*;
import org.hibernate.cfg.*;

public class StoreData {
public static void main(String[] args) {
    Session session=new Configuration().configure("hibernate.cfg.xml")
                            .buildSessionFactory().openSession();
    Transaction t=session.beginTransaction();

    Answer ans1=new Answer();
    ans1.setAnswername("java is a programming language");
    ans1.setPostedBy("Ravi Malik");

    Answer ans2=new Answer();
    ans2.setAnswername("java is a platform");
    ans2.setPostedBy("Sudhir Kumar");
```

```
    Answer ans3=new Answer();
    ans3.setAnswername("Servlet is an Interface");
    ans3.setPostedBy("Jai Kumar");

    Answer ans4=new Answer();
    ans4.setAnswername("Servlet is an API");
    ans4.setPostedBy("Arun");

    ArrayList<answer> list1=new ArrayList<answer>();
    list1.add(ans1);
    list1.add(ans2);

    ArrayList<answer> list2=new ArrayList<answer>();
    list2.add(ans3);
    list2.add(ans4);

    Question question1=new Question();
    question1.setQname("What is Java?");
    question1.setAnswers(list1);

    Question question2=new Question();
    question2.setQname("What is Servlet?");
    question2.setAnswers(list2);

    session.persist(question1);
    session.persist(question2);

    t.commit();
    session.close();
    System.out.println("success");

}
}
 </answer></answer></answer></answer>
```

آموزش نحوه دریافت fetch اطلاعات از List

در این بخش، ما از زبان HQL برای خواندن و دریافت اطلاعات کلاس Question Class از جمله رکوردهای answers پرداخته ایم. در این مثال، دستورات ما اطلاعات را از دو جدول مختلف که به صورت مستقل برنامه ای (functional dependent) با هم در ارتباط هستند، استخراج می کند.

در این کد ما نحوه چاپ خروجی اشیای کلاس answer را تعیین کرده ایم و متد ()to string که از کلاس Answer اطلاعات فیلدهای answername و poster name را نادیده می گیریم. بنابراین کد ما مقایسه answer name و postername را در خروجی به جای refrence چاپ می کند.

فایل FetchData.java.

```
package com.javatpoint;

import java.util.*;

import org.hibernate.*;
import org.hibernate.cfg.*;

public class FetchData {
public static void main(String[] args) {

    Session session=new Configuration().configure("hibernate.cfg.xml")
                        .buildSessionFactory().openSession();

    Query query=session.createQuery("from Question");
    List<question> list=query.list();

    Iterator<question> itr=list.iterator();
    while(itr.hasNext()){
        Question q=itr.next();
        System.out.println("Question Name: "+q.getQname());

        //printing answers
        List<answer> list2=q.getAnswers();
        Iterator<answer> itr2=list2.iterator();
        while(itr2.hasNext()){
            System.out.println(itr2.next());
        }

    }
    session.close();
    System.out.println("success");

}
}
            </answer></answer></question></question>
```

آموزش آدرس دهی متغیرهای Set در مجموعه های Hibernate

اگر کلاس Persistent Class شما شامل متغیرهای Set objects می باشد، می توانید از المنت Set در فایل آدرس دهی mapping file برای map متغیر Set استفاده کنید.

المنت Set نیازی به index گذاری ندارد. متغیرهای Set و List کاملا شبیه هم هستند، با این تفاوت که متغیرهای Set فقط مقادیر یکتا و غیر تکراری (Unique) را در اعضای خود نگهداری می کند.

به وسیله کد زیر، می توانید یک متغیر Set را تعیین و پیاده سازی کنید:

```
<class name="com.javatpoint.Question" table="q102">
        ...
           <set name="answers" table="ans102">
           <key column="qid"></key>
```

```
        <element column="answer" type="string"></element>
        </set>

      ...
</class>
```

مثال عملی آموزش آدرس دهی متغیر Set در مجموعه ها Hibernate

در مثال این درس قصد داریم تا نحوه آدرس دهی mapping متغیرهای Set را به طور کامل در Hibernate آموزش دهیم.

در این مثال، متغیر Set مقادیر (Values) را نگهداری می کند نه رفرنس های entity. به همین دلیل است که ما از element به جای one-to-many استفاده می کنیم.

1. ایجاد کلاس Persistent Class

در مرحله اول اقدام به ایجاد کلاس Persistent می کنیم که خواص لازم برای کلاس از جمله Set را تعیین می کند:

```
package com.javatpoint;

import java.util.Set;

public class Question {
private int id;
private String qname;
private Set<string> answers;

//getters and setters

}
            </string>
```

2. ایجاد فایل آدرس دهی mapping file برای فایل Persistent

در این مرحله، ما کلاس question.hbm.xml را برای تعیین list ایجاد می کنیم.

```
<!--?xml version='1.0' encoding='UTF-8'?-->


<hibernate-mapping>
 <class name="com.javatpoint.Question" table="q102">
   <id name="id">
     <generator class="increment"></generator>
   </id>
```

```
   <property name="qname"></property>

   <set name="answers" table="ans102">
     <key column="qid"></key>
     <element column="answer" type="string"></element>
   </set>

 </class>

</hibernate-mapping>
```

3. ایجاد فایل تنظیمات Configuration file

در مرحله بعدی، فایل تنظیمات برنامه را ایجاد می کنیم که اطلاعات لازم درباره پایگاه داده و فایل آدرس دهی mapping file را شامل می شود.

```
<!--?xml version='1.0' encoding='UTF-8'?-->


<!-- Generated by MyEclipse Hibernate Tools.                    -->
<hibernate-configuration>

    <session-factory>
        <property name="hbm2ddl.auto">update</property>
        <property
name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
        <property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
        <property name="connection.username">system</property>
        <property name="connection.password">oracle</property>
        <property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
    <mapping resource="question.hbm.xml">
    </mapping></session-factory>

</hibernate-configuration>
```

4. ایجاد کلاس لازم جهت نگهداری اطلاعات برنامه:

در کلاس زیر کد لازم جهت نگهداری اطلاعات کلاس question را تعیین کرده ایم:

```
package com.javatpoint;

import java.util.ArrayList;

import org.hibernate.*;
import org.hibernate.cfg.*;

public class StoreData {
```

```
 public static void main(String[] args) {
    Session session=new Configuration().configure("hibernate.cfg.xml")
                        .buildSessionFactory().openSession();
    Transaction t=session.beginTransaction();


    HashSet<string> set1=new HashSet<string>();
    set1.add("java is a programming language");
    set1.add("java is a platform");

    HashSet<string> set2=new HashSet<string>();
    set2.add("Servlet is an Interface");
    set2.add("Servlet is an API");

    Question question1=new Question();
    question1.setQname("What is Java?");
    question1.setAnswers(set1);

    Question question2=new Question();
    question2.setQname("What is Servlet?");
    question2.setAnswers(set2);

    session.persist(question1);
    session.persist(question2);

    t.commit();
    session.close();
    System.out.println("success");
 }
}
            </string></string></string></string>
```

آموزش نحوه خواندن و دریافت fetch اطلاعات:

در این بخش نیز از زبان HQL برای دریافت اطلاعات کلاس Question از جمله رکوردهای answers استفاده می کنیم. در این مورد، برنامه اطلاعات را از دو جدول که به صورت کد مستقل functional dependent به هم متصل هستند، استخراج می کند:

```
package com.javatpoint;

import java.util.*;

import org.hibernate.*;
import org.hibernate.cfg.*;

public class FetchData {
public static void main(String[] args) {

    Session session=new Configuration().configure("hibernate.cfg.xml")
                        .buildSessionFactory().openSession();
```

```
    Query query=session.createQuery("from Question");
    List<question> list=query.list();

    Iterator<question> itr=list.iterator();
    while(itr.hasNext()){
        Question q=itr.next();
        System.out.println("Question Name: "+q.getQname());

        //printing answers
        Set<string> set=q.getAnswers();
        Iterator<string> itr2=set.iterator();
        while(itr2.hasNext()){
            System.out.println(itr2.next());
        }

    }
    session.close();
    System.out.println("success");

}
}
            </string></string></question></question>
```

آموزش آدرس دهی متغیرهای Set با روش One to Many در Hibernate

اگر کلاس Persistent Class شما دارای اشیای Set ای است که محتوی رفرنس های entity می باشند، بایستی از روش one-to-many برای آدرس دهی یا map المنت Set استفاده کنیم. شما همچنین می توانید این list object را با المنت Set آدرس دهی کنید.

ابتدا بیایید نگاهی به کد کلاس Persistent Class که دارای اشیای set objects می باشند، بیاندازیم. در این کلاس، هر question می تواند دارای چندین answer باشد که هر answer نیز دارای اطلاعات مختص به خود می باشد .برای همین است که ما از المنت Set برای جایگزینی مجموعه ای از جواب ها (answers) استفاده کرده ایم، به صورت زیر:

```
package com.javatpoint;

import java.util.List;

public class Question {
private int id;
private String qname;
private Set<answer> answers;
//getters and setters
```

```
}
            </answer>
```

خود کلاس Answer Class حاوی اطلاعات مختص به خود مثل id ، answers، postedBy و ... می باشد، به صورت زیر:

```
package com.javatpoint;

public class Answer {
private int id;
private String answername;
private String postedBy;
//getters and setters

}
}
```

کلاس Question Class حاوی Set objects می باشد که خود شامل رفرنس های entity مثل Answer class object می باشد. در چنین موردی، ما از روش one-to-many برای آدرس دهی map این شی به صورت زیر استفاده می کنیم:

```
<set name="answers" cascade="all">
          <key column="qid"></key>
          <one-to-many class="com.javatpoint.Answer">
</one-to-many></set>
```

مثال عملی آدرس دهی Set در مجموعه های Hibernate با استفاده از روش one to many

مثال این درس کاملا شبیه مثال درس one to many bag است با این تفاوت که در کدها، همه bag را به Set در فایل hbm تغیر داده ایم. همچنین در کلاس Store مقدار ArrayList را به HashSet تغییر می دهیم.

آموزش آدرس دهی Map در مجموعه های Hibernate با استفاده از فایل xml

Hibernateبه شما امکان آدرس دهی المنت Map را به وسیله RD MS می دهد .همان طور که می دانید المنت های map وlist ، ایندکس گذاری شده یا index-basesdهستند. در این موارد، ستون index به عنوان کلید key و ستون elementبه عنوان مقدار یا Value عمل می کند.

مثال عملی آدرس دهی Map در مجموعه های Hibernate با استفاده از فایل xml

برای آدرس دهی المنت map بایستی صفحات زیر را در برنامه خود ایجاد کنید:

- Question.java
- question.hbm.xml
- hibernate.cfg.xml
- StoreTest.java
- FetchTest.java

کد هر یک از فایل های فوق به صورت زیر تعیین می شوند:

فایلQuestion.java

```
package com.javatpoint;

import java.util.Map;

public class Question {
private int id;
private String name,username;
private Map<string,string> answers;

public Question() {}
public Question(String name, String username, Map<string, string=""> answers)
{
    super();
    this.name = name;
    this.username = username;
    this.answers = answers;
}
public int getId() {
    return id;
}
public void setId(int id) {
    this.id = id;
}
```

```
public String getName() {
    return name;
}
public void setName(String name) {
    this.name = name;
}
public String getUsername() {
    return username;
}
public void setUsername(String username) {
    this.username = username;
}
public Map<string, string=""> getAnswers() {
    return answers;
}
public void setAnswers(Map<string, string=""> answers) {
    this.answers = answers;
}
}
            </string,></string,></string,></string,string>
```

فایل question.hbm.xml

```
<!--?xml version='1.0' encoding='UTF-8'?-->


<hibernate-mapping>

<class name="com.javatpoint.Question" table="question736">
<id name="id">
<generator class="native"></generator>
</id>
<property name="name"></property>
<property name="username"></property>

<map name="answers" table="answer736" cascade="all">
<key column="questionid"></key>
<index column="answer" type="string"></index>
<element column="username" type="string"></element>
</map>
</class>

</hibernate-mapping>
```

فایل hibernate.cfg.xml

```
<!--?xml version='1.0' encoding='UTF-8'?-->


<!-- Generated by MyEclipse Hibernate Tools.                    -->
<hibernate-configuration>

    <session-factory>
```

```
        <property name="hbm2ddl.auto">update</property>
        <property
name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
        <property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
        <property name="connection.username">system</property>
        <property name="connection.password">oracle</property>
        <property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>

    <mapping resource="question.hbm.xml">
    </mapping></session-factory>

</hibernate-configuration>
```

فایل StoreTest.java

```
package com.javatpoint;

import java.util.HashMap;
import org.hibernate.*;
import org.hibernate.cfg.*;
public class StoreTest {
public static void main(String[] args) {
Session session=new
Configuration().configure().buildSessionFactory().openSession();
Transaction tx=session.beginTransaction();

HashMap<string,string> map1=new HashMap<string,string>();
map1.put("java is a programming language","John Milton");
map1.put("java is a platform","Ashok Kumar");

HashMap<string,string> map2=new HashMap<string,string>();
map2.put("servlet technology is a server side programming","John Milton");
map2.put("Servlet is an Interface","Ashok Kumar");
map2.put("Servlet is a package","Rahul Kumar");

Question question1=new Question("What is java?","Alok",map1);
Question question2=new Question("What is servlet?","Jai Dixit",map2);

session.persist(question1);
session.persist(question2);

tx.commit();
session.close();
System.out.println("successfully stored");
}
}
            </string,string></string,string></string,string></string,string>
```

فایل FetchTest.java

```
package com.javatpoint;
import java.util.*;
import org.hibernate.*;
```

```
import org.hibernate.cfg.*;
public class FetchTest {
public static void main(String[] args) {
 Session session=new
Configuration().configure().buildSessionFactory().openSession();

 Query query=session.createQuery("from Question ");
 List<question> list=query.list();

 Iterator<question> iterator=list.iterator();
 while(iterator.hasNext()){
  Question question=iterator.next();
  System.out.println("question id:"+question.getId());
  System.out.println("question name:"+question.getName());
  System.out.println("question posted by:"+question.getUsername());
  System.out.println("answers.....");
  Map<string,string> map=question.getAnswers();
  Set<map.entry<string,string>> set=map.entrySet();

  Iterator<map.entry<string,string>> iteratoranswer=set.iterator();
  while(iteratoranswer.hasNext()){
   Map.Entry<string,string>
entry=(Map.Entry<string,string>)iteratoranswer.next();
   System.out.println("answer name:"+entry.getKey());
   System.out.println("answer posted by:"+entry.getValue());
  }
 }
session.close();
}
}
            </string,string></string,string></map.entry<string,string></map.e
ntry<string,string>
```

آموزش آدرس دهی Many to Many Mapping در Hibernate با مثال mapو استفاده از فایل xml

شما می توانید ارتباط many to many را با استفاده از المنت هایset ، bag، mapو غیره انجام دهید. در این درس قصد داریم تا از المنت map برای آدرس دهی many to many استفاده کنیم. در این حالت، 3 جدول ایجاد خواهد شد که به بررسی آن ها خواهیم پرداخت.

مثال عملی آموزش آدرس دهی Many to Many در Hibernate

برای اجرای مثال نیاز دارید تا صفحه های زیر را در پروژه خود اضافه کنید:

- Question.java
- User.java

- question.hbm.xml
- User.hbm.xml
- hibernate.cfg.xml
- StoreTest.java
- FetchTest.java

کد هر یک از فایل های فوق به صورت زیر خواهد بود:

فایلQuestion.java

```
package com.javatpoint;

import java.util.Map;

public class Question {
private int id;
private String name;
private Map<string,user> answers;

public Question() {}
public Question(String name, Map<string, user=""> answers) {
    super();
    this.name = name;
    this.answers = answers;
}
public int getId() {
    return id;
}
public void setId(int id) {
    this.id = id;
}
public String getName() {
    return name;
}
public void setName(String name) {
    this.name = name;
}
public Map<string, user=""> getAnswers() {
    return answers;
}
public void setAnswers(Map<string, user=""> answers) {
    this.answers = answers;
}


}
```

```
            </string,></string,></string,></string,user>
```

فایلUser.java

```
package com.javatpoint;

public class User {
private int id;
private String username,email,country;

public User() {}
public User(String username, String email, String country) {
    super();
    this.username = username;
    this.email = email;
    this.country = country;
}
public int getId() {
    return id;
}

public void setId(int id) {
    this.id = id;
}

public String getUsername() {
    return username;
}

public void setUsername(String username) {
    this.username = username;
}

public String getEmail() {
    return email;
}

public void setEmail(String email) {
    this.email = email;
}

public String getCountry() {
    return country;
}

public void setCountry(String country) {
    this.country = country;
}
}
```

فایلquestion.hbm.xml

```
<!--?xml version='1.0' encoding='UTF-8'?-->
```

```
<hibernate-mapping>
<class name="com.javatpoint.Question" table="question738">
<id name="id">
<generator class="native"></generator>
</id>
<property name="name"></property>

<map name="answers" table="answer738" cascade="all">
<key column="questionid"></key>
<index column="answer" type="string"></index>
<many-to-many class="com.javatpoint.User" column="userid"></many-to-many>
</map>
</class>

</hibernate-mapping>
```

فایل User.hbm.xml

```
<!--?xml version='1.0' encoding='UTF-8'?-->


<hibernate-mapping>
<class name="com.javatpoint.User" table="user738">
<id name="id">
<generator class="native"></generator>
</id>
<property name="username"></property>
<property name="email"></property>
<property name="country"></property>
</class>

</hibernate-mapping>
```

فایل hibernate.cfg.xml

```
<!--?xml version='1.0' encoding='UTF-8'?-->


<!-- Generated by MyEclipse Hibernate Tools.                    -->
<hibernate-configuration>

  <session-factory>
   <property name="hbm2ddl.auto">update</property>
   <property name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
   <property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
   <property name="connection.username">system</property>
   <property name="connection.password">oracle</property>
   <property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>

    <mapping resource="question.hbm.xml">
    <mapping resource="user.hbm.xml">
    </mapping></mapping></session-factory>
```

```
</hibernate-configuration>
```

فایلStoreTest.java

```
package com.javatpoint;

import java.util.HashMap;
import org.hibernate.*;
import org.hibernate.cfg.*;
public class StoreTest {
public static void main(String[] args) {
Session session=new
Configuration().configure().buildSessionFactory().openSession();
Transaction tx=session.beginTransaction();

HashMap<string,user> map1=new HashMap<string,user>();
map1.put("java is a programming language",
new User("John Milton","john@gmail.com","usa"));

map1.put("java is a platform",
new User("Ashok Kumar","ashok@gmail.com","india"));

HashMap<string,user> map2=new HashMap<string,user>();
map2.put("servlet technology is a server side programming",

new User("John Milton","john@gmail.com","usa"));
map2.put("Servlet is an Interface",
new User("Ashok Kumar","ashok@gmail.com","india"));

Question question1=new Question("What is java?",map1);
Question question2=new Question("What is servlet?",map2);

session.persist(question1);
session.persist(question2);

tx.commit();
session.close();
System.out.println("successfully stored");
}
}
            </string,user></string,user></string,user></string,user>
```

فایلFetchTest.java

```
package com.javatpoint;
import java.util.*;
import org.hibernate.*;
import org.hibernate.cfg.*;
public class FetchTest {
public static void main(String[] args) {
 Session session=new
Configuration().configure().buildSessionFactory().openSession();
 Query query=session.createQuery("from Question ");
 List<question> list=query.list();
```

```
 Iterator<question> iterator=list.iterator();
 while(iterator.hasNext()){
  Question question=iterator.next();
  System.out.println("question id:"+question.getId());
  System.out.println("question name:"+question.getName());
  System.out.println("answers.....");
  Map<string,user> map=question.getAnswers();
  Set<map.entry<string,user>> set=map.entrySet();

  Iterator<map.entry<string,user>> iteratoranswer=set.iterator();
  while(iteratoranswer.hasNext()){
   Map.Entry<string,user>
entry=(Map.Entry<string,user>)iteratoranswer.next();
   System.out.println("answer name:"+entry.getKey());
   System.out.println("answer posted by.........");
   User user=entry.getValue();
   System.out.println("username:"+user.getUsername());
   System.out.println("user emailid:"+user.getEmail());
   System.out.println("user country:"+user.getCountry());
  }
 }
session.close();
}
}
       </string,user></string,user></map.entry<string,user></map.entry<string
,user></string,user></question></question>
```

آموزش روابط دو طرفه Bidirectional Assocation در Hibernate

روابط دو طرفه یا Bidirectional Assocation به شما امکان خواندن و دریافت fetch اطلاعات اشیای وابسته یا object dependent را از دو طرف می دهد. در چنین مواردی، رفرنس هایی مربوط به هر یک از کلاس ها در کلاس دیگر وجود دارد.

برای روشن شدن مسئله به بررسی ارتباط Employee و Address می پردازیم. در این حالت بایستی کلاس Employee Classیک رفرنس به کلاس Address و کلاس Addressیک رفرنس به کلاس Employee داشته باشد.

علاوه بر این، شما ارتباطات one-to-many و one-to-one را نیز در فایل آدرس دهی mapping file ایجاد کرده اید، که در اصطلاح به آن ها ارتباطات دو طرفه یا Bidirectional Assocation می گویند.

برای درک بهتر این مطلب، به درس های آموزش رابط های one-to-one و one-to-many در همین مبحث درسی بروید.

آموزش کاربرد Hibernate Lazy Collection

استراتژی Lazy Collection خواندن حداقل اطلاعات لازم جهت بالا بردن کارایی سیستم) فقط بر حسب نیاز اشیای فرزند (Child objects) برنامه را لود می کند. این کار برای بالا بردن کارایی یا Perfomance سیستم استفاده می شود. از Hibernate 30 به بعد، قابلیت Lazy Collection به صورت پیش فرض فعال است.

برای استفاده از امکان Lazy Collection در برنامه هایHibernate ، بایستی مقدار خاصیت lazy="true" را در Collection خود قرار دهید. همان طور که اشاره شد، این امکان به صورت پیش فرض در Hibernate 30 به بعد فعال است و اصولا نیاز به انجام کار فوق نیست. اما اگر مقدار خاصیت lazy را بر روی falseقرار دهید، آنگاه برنامه کلید اشیای فرزند Child objects را در هنگام اجرا لود می کند که باعث کاهش شدید کارایی یا Perfomanceسیستم مخصوصا در حجم زیاد داده ها می شود. به صورت زیر می توانید مقدار خاصیت lazy را در Collectionخود بر روی true تنظیم کنید:

```
<list name="answers" lazy="true">
          <key column="qid"></key>
          <index column="type"></index>
          <one-to-many class="com.javatpoint.Answer">
</one-to-many></list>
```

آموزش آدرس دهی Component Mapping در Hibernate

در استراتژی Component Mapping ما هر شی وابسته یا dependent object را به عنوان یک Componentآدرس دهی می کنیم.

یک Component یک شی یا object است که به صورت مقدار یا Value نگهداری می شود تا refrence. این روش معمولا زمانی استفاده می شود که dependent object دارای کلید اصلی Primary key نیست.

همچنین Component Mapping برای انجام عمل ترکیب یا Composition که دارای ارتباط HAS-(A relationبه کار رفته و به همین دلیل Component نامگذاری شده است.

بیایید به کد کلاس هایی که دارای ارتباط HAA-A relation هستند، نگاهی بکنیم:

```
package com.javatpoint;

public class Address {
private String city,country;
private int pincode;

//getters and setters
}
```

```
package com.javatpoint;
public class Employee {
private int id;
private String name;
private Address address;//HAS-A

//getters and setters
}
```

در کد فوق، addressیک شی وابسته یا dependent object است. چهارچوب کاری Hibernateامکان آدرس دهی یا map یک شی وابسته dependent object به صورت یک Componentفراهم کرده است.

در مثال زیر به آموزش آدرس دهی یا map یک شی وابسته dependent object در فایل آدرس دهی mapping fileپرداخته ایم:

```
<class name="com.javatpoint.Employee" table="emp177">
<id name="id">
<generator class="increment"></generator>
</id>
<property name="name"></property>

<component name="address" class="com.javatpoint.Address">
<property name="city"></property>
<property name="country"></property>
<property name="pincode"></property>
</component>

</class>
```

خروجی مثال هم به صورت زیر است:

ID	NAME	CITY	COUNTRY	PINCODE
1	amit	gzb	india	221233
2	Vijay	noida	india	224123

آموزش آدرس دهی Association Mapping با روش one-to-one در Hibernate

شما می توانید روش آدرس دهی one to one mapping را از دو راه انجام دهید:

- به وسیله المنت.many-to-one
- به وسیله المنت.many-to-one

در این درس، قصد داریم روش اجرای آدرس دهی one to one mapping را به وسیله المنت-to– many one آموزش دهیم. در چنین موردی، یک کلید خارجی foreign key در جدول اصلی برنامه ایجاد می شود.

در این مثال، هر employee فقط یک address داشته و هر address فقط متعلق به یک employee است.

همچنین در این مثال ما از استراتژی روابط دو طرفه یا Bridirectional Association استفاده می کنیم.

برای اجرای مثال، مراحل زیر را به ترتیب انجام دهید:

1. ایجاد کلاس Persistent Class بر آدرس دهی: one to one mapping

در مثال دو کلاس Persistent Class با نام های Employee.java و Address.java وجود دارد.

کلاس Employee حاوی یک رفرنس به کلاس Address می باشد و برعکس.

فایلEmployee.java

```
package com.javatpoint;

public class Employee {
private int employeeId;
private String name,email;
private Address address;
//setters and getters
}
```

فایلAddress.java

```
package com.javatpoint;

public class Address {
private int addressId;
private String addressLine1,city,state,country;
private int pincode;
private Employee employee;
//setters and getters
}
```

2. ایجاد فایل آدرس دهی Mapping files جهت کلاس Persistent
مثال دارای دو فایل آدرس دهی mapping file به نام های employee.hbm.xml و address.hbm.xml است که کد آن ها در ادامه قرار داده شده است.

فایل employee.hbm.xml

```
<!--?xml version='1.0' encoding='UTF-8'?-->


          <hibernate-mapping>
          <class name="com.javatpoint.Employee" table="emp211">
          <id name="employeeId">
          <generator class="increment"></generator>
          </id>
          <property name="name"></property>
          <property name="email"></property>

          <many-to-one name="address" unique="true" cascade="all"></many-to-
one>
          </class>

</hibernate-mapping>
```

در این فایل آدرس دهی mapping file ما از المنت many –to- one با خاصیت "Unique="true جهت آدرس دهی one to many mapping استفاده کرده ایم:

فایل address.hbm.xml

کد زیر فایل ساده آدرس دهی mapping file جهت کلاس Address را نشان می دهد:

```
<!--?xml version='1.0' encoding='UTF-8'?-->


          <hibernate-mapping>
          <class name="com.javatpoint.Address" table="address211">
          <id name="addressId">
          <generator class="increment"></generator>
          </id>
```

```
        <property name="addressLine1"></property>
        <property name="city"></property>
        <property name="state"></property>
        <property name="country"></property>

        </class>

</hibernate-mapping>
```

3. ایجاد فایل تنظیمات Configuration file

فایل تنظیمات Configuration file حاوی اطلاعات لازم جهت کار با پایگاه داده و mpping file است:

فایل hibernate.cfg.xml

```
<!--?xml version='1.0' encoding='UTF-8'?-->


<!-- Generated by MyEclipse Hibernate Tools.                    -->
<hibernate-configuration>

    <session-factory>
        <property name="hbm2ddl.auto">update</property>
        <property
name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
        <property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
        <property name="connection.username">system</property>
        <property name="connection.password">oracle</property>
        <property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
    <mapping resource="employee.hbm.xml">
    <mapping resource="address.hbm.xml">
    </mapping></mapping></session-factory>

</hibernate-configuration>
```

4. ایجاد کلاس های لازم جهت استخراج fetch و ذخیره اطلاعات:

فایل Store.java

```
package com.javatpoint;
import org.hibernate.cfg.*;
import org.hibernate.*;

public class Store {
public static void main(String[] args) {
    Configuration cfg=new Configuration();
    cfg.configure("hibernate.cfg.xml");
    SessionFactory sf=cfg.buildSessionFactory();
```

```
    Session session=sf.openSession();
    Transaction tx=session.beginTransaction();

    Employee e1=new Employee();
    e1.setName("Ravi Malik");
    e1.setEmail("ravi@gmail.com");

    Address address1=new Address();
    address1.setAddressLine1("G-21,Lohia nagar");
    address1.setCity("Ghaziabad");
    address1.setState("UP");
    address1.setCountry("India");
    address1.setPincode(201301);


    e1.setAddress(address1);
    address1.setEmployee(e1);

    session.persist(e1);
    tx.commit();

    session.close();
    System.out.println("success");
}
}
```

فایل Fetch.java

```
package com.javatpoint;
import java.util.Iterator;
import java.util.List;
import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class Fetch {
public static void main(String[] args) {
    Configuration cfg=new Configuration();
    cfg.configure("hibernate.cfg.xml");
    SessionFactory sf=cfg.buildSessionFactory();
    Session session=sf.openSession();

    Query query=session.createQuery("from Employee e");
    List<employee> list=query.list();

    Iterator<employee> itr=list.iterator();
    while(itr.hasNext()){
     Employee emp=itr.next();
     System.out.println(emp.getEmployeeId()+" "+emp.getName()+"
"+emp.getEmail());
     Address address=emp.getAddress();
     System.out.println(address.getAddressLine1()+" "+address.getCity()+" "+
                                address.getState()+" "+address.getCountry());
```

```
    }

    session.close();
    System.out.println("success");
}
}

            </employee></employee>
```

آموزش آدرس دهی Association Mapping با روش one-to-one در Hibernate – بخش دوم

همانطور که در درس قبل مطرح کردیم، دو راه برای آدرس دهی one to one mapping در Hibernate وجود دارد که عبارتند از:

- استفاده از المنت many-to-one.
- استفاده از المنت one-to-one.

در این درس قصد داریم روش اجرای آدرس دهی one to one mapping را با استفاده از المنت one-to-one آموزش دهیم. در چنین موردی، هیچ کلید خارجی foreign key به جدول اصلی برنامه اضافه نمی شود. در این مثال، هر employee فقط یک آدرس (Address) داشته و هر Address نیز فقط متعلق یک employee است. در این مثال، از استراتژی روابط دو طرفه یا Bidirectional Association استفاده خواهیم کرد.

برای اجرای مثال، مراحل زیر را به ترتیب انجام دهید:

1. ایجاد کلاس Persistent Class برای آدرس دهی one to one mapping

در مثال دو کلاس Employee.java و Address.java به عنوان کلاس های Persistent قرار دارند. کلاس Employee حاوی یک رفرنس به کلاس Address می باشد و برعکس.

فایل Employee.java

```
package com.javatpoint;

public class Employee {
private int employeeId;
private String name,email;
private Address address;
//setters and getters
```

```
}
```

فایلAddress.java

```
package com.javatpoint;

public class Address {
private int addressId;
private String addressLine1,city,state,country;
private int pincode;
private Employee employee;
//setters and getters
}
```

2. ایجاد فایل آدرس دهی Mapping file جهت کلاس Persistent

مثال دارای دو فایل آدرس دهی mapping file با نام های employee.hbm.xml و

address.hbm.xmlاست که کد آن ها در ادامه قرار داده شده است.

فایلemployee.hbm.xml

```
<!--?xml version='1.0' encoding='UTF-8'?-->


          <hibernate-mapping>
          <class name="com.javatpoint.Employee" table="emp212">
          <id name="employeeId">
          <generator class="increment"></generator>
          </id>
          <property name="name"></property>
          <property name="email"></property>

          <one-to-one name="address" cascade="all"></one-to-one>
          </class>

</hibernate-mapping>
```

در این فایل آدرس دهی mapping file ها در هر دو فایل از المنت one-to-one برای آدرس دهی یا

mappingاستفاد ه کرده ایم:

کد زیر فایل ساده آدرس دهی mapping file را برای کلاس Address نشان می دهد. اما نکته مهم وجود

کلاس سازنده یا generator است. در اینجا ما داریم از یک کلاس سازنده خارجی یا foreign generator

classاستفاده می کنیم که بر اساس کلید اصلی کلاس Employee کار می کند.

فایل address.hbm.xml

```
<!--?xml version='1.0' encoding='UTF-8'?-->
<  !DOC TYPE hibernate-mapping PUBLIC
          "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
          "hibernate.sourceforge.net/hibernate-mapping-3.0.dtd" >

           <hibernate-mapping>
          <class name="com.javatpoint.Address" table="address212">
          <id name="addressId">
          <generator class="foreign">
          <param name="property">employee
          </generator>
          </id>
          <property name="addressLine1"></property>
          <property name="city"></property>
          <property name="state"></property>
          <property name="country"></property>

          <one-to-one name="employee"></one-to-one>
          </class>

</hibernate-mapping>
```

3. ایجاد فایل تنظیمات Congiguration file

فایل تنظیمات Configuration file اطلاعات لازم جهت کار با database و فایل آدرس دهی mapping fileمی باشد.

فایل hibernate.cfg.xml

```
<!--?xml version='1.0' encoding='UTF-8'?-->
< !DOC TYPE hibernate-configuration PUBLIC
          "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
          "hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<!-- Generated by MyEclipse Hibernate Tools.                    -->
<hibernate-configuration>

    <session-factory>
        <property name="hbm2ddl.auto">update</property>
        <property
name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
        <property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
        <property name="connection.username">system</property>
        <property name="connection.password">oracle</property>
        <property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
    <mapping resource="employee.hbm.xml">
```

```
    <mapping resource="address.hbm.xml">
    </mapping></mapping></session-factory>

</hibernate-configuration>
```

ایجاد کلاس های لازم برای دریافت و نگهداری اطلاعات:

فایلStore.java

```
package com.javatpoint;
import org.hibernate.cfg.*;
import org.hibernate.*;

public class Store {
public static void main(String[] args) {
    Configuration cfg=new Configuration();
    cfg.configure("hibernate.cfg.xml");
    SessionFactory sf=cfg.buildSessionFactory();
    Session session=sf.openSession();
    Transaction tx=session.beginTransaction();

    Employee e1=new Employee();
    e1.setName("Ravi Malik");
    e1.setEmail("ravi@gmail.com");

    Address address1=new Address();
    address1.setAddressLine1("G-21,Lohia nagar");
    address1.setCity("Ghaziabad");
    address1.setState("UP");
    address1.setCountry("India");
    address1.setPincode(201301);


    e1.setAddress(address1);
    address1.setEmployee(e1);

    session.persist(e1);
    tx.commit();

    session.close();
    System.out.println("success");
}
}
```

فایلFetch.java

```
package com.javatpoint;
import java.util.Iterator;
import java.util.List;
```

```
import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class Fetch {
public static void main(String[] args) {
    Configuration cfg=new Configuration();
    cfg.configure("hibernate.cfg.xml");
    SessionFactory sf=cfg.buildSessionFactory();
    Session session=sf.openSession();

    Query query=session.createQuery("from Employee e");
    List<employee> list=query.list();

    Iterator<employee> itr=list.iterator();
    while(itr.hasNext()){
     Employee emp=itr.next();
     System.out.println(emp.getEmployeeId()+" "+emp.getName()+"
"+emp.getEmail());
     Address address=emp.getAddress();
     System.out.println(address.getAddressLine1()+" "+address.getCity()+" "+
        address.getState()+" "+address.getCountry());
    }

    session.close();
    System.out.println("success");
}
}
            </employee></employee>
```

آموزش مدیریت تراکنش ها Tranaction Managment در Hibernate

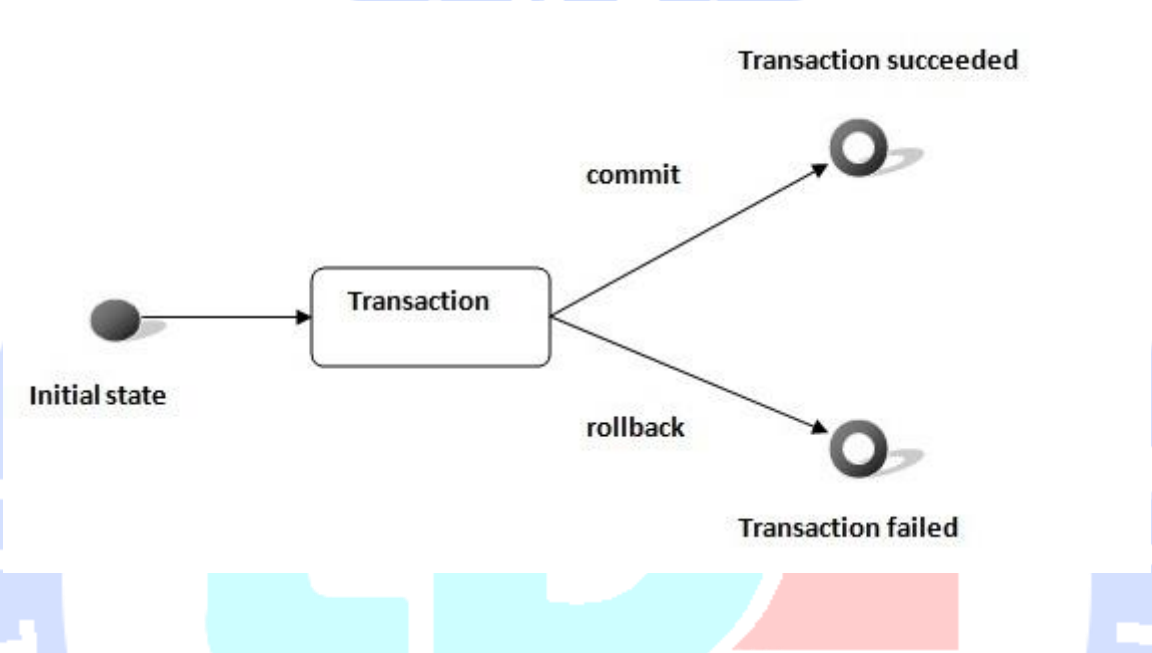
یک تراکنش (transaction) به طور ساده عبارت است از یک واحد مستقل کار. در چنین موردی، وقتی یک مرحله از تراکنش ناموفق اجرا شود، بایستی کل تراکنش لغو گردد، وگرنه نتیجه ناقص در بر خواهد داشت (به این اصل در اصطلاح atomicity یا یکپارچگی می گویند.)

یک تراکنش را می توان به وسیله چهار خاصیت یا Properties مهم که در اختصار به آن ها ACFO می گویند، توصیف کرد. این خواص عبارتند از:

- Atomicity یا یکپارچگی.
- Consistency یا ثبات.

- Isolationیا ایزوله بودن
- Durabilityیا استمرار.

شکل زیر، چارت عملکرد یک تراکنش را نشان می دهد:



رابط کاربری تراکنش یا Transaction Interface در Hibernate

در چهارچوب کاریHibernate ، ما مفهومی داریم به نام رابط کاربری تراکنش یا Transaction Interface که یک واحد کاری مستقل را جهت تراکنش تعیین می کند. این رابط کاربری خود تراکنش را از پروسه اجرا و پیاده سازی آن جدا و مستقل می کند.

یک تراکنش رابطه مستقیمی با مفهوم Session در برنامه داشته و با فراخوانی متد Session.begin Transaction()شروع می شود. سایر متدهای مرتبط با تراکنش در Hibernate عبارتند از:

- متد Void begin() که یک تراکنش جدید را شروع می کند.
- متد Void Commit() یک واحد کاری تراکنش را به اتمام می رساند مگر این که در وضعیت FlashMode.NEVERباشیم.

- متد( )Void rollback ، تراکنش را مجبور می کند تا لغو شده و اثر خود را کنسل کند) در اصطلاح rollbackشود.)
- متد (int seconds) Void SetTimeout ، این متد یک محدوده زمانی اتمام یا Transaction Timeout را برای تراکنش تعیین می کند که بایستی تراکنش در محدوده ثانیه تعیین شده شروع شده و پایان یابد.
- متد( )Boolean is Alive ، چک می کند که آیا تراکنش هنوز فعال است یا خیر.
- ( )، چک می کند آیا تراکنش به صورت درست اجرا شده است یا خیر.
- متد( )Boolean was Rolled Back ، چک می کند آیا تراکنش به صورت صحیح لغو شده و اثر ناقص آن خنثی شده یا خیر.
- متدr+ Void ، یک عملیات را جهت هماهنگ سازی rollback کاربر تعیین می کند.

مثال و کد عملی آموزش مدیریت تراکنش ها Tranaction Managment در Hibernate

در Hibernate زمانی که در یک تراکنش خطایی رخ داده و عملیات ناقص اجرا می شود ، بهتر است تا کل تراکنش و اثرات آن لغو گردند ، تا اولا در ثبت اطلاعات مشکلی به وجود نیامده و دوما منابع درگیر با تراکنش سیستم آزاد شوند . در کد مثال زیر ، نحوه مدیریت کامل یک تراکنش را در Hibernate آموزش داده ایم:

```
Session session = null;
Transaction tx = null;

try {
session = sessionFactory.openSession();
tx = session.beginTransaction();
//some action

tx.commit();

}catch (Exception ex) {
ex.printStackTrace();
tx.rollback();
}
finally {session.close();}
```

آموزش زبان کار با پایگاه داده HQL در Hibernate

زبان کار با پایگاه داده در Hibernate یا Hibernate Query Language که ساختار HQL نامیده می شود، کاملا شبیه زبان SQL بوده با این تفاوت که زبان HQL مبتنی بر جدول پایگاه داده نیست. به جای استفاده از نام جدول در HQL ما از نام کلاس استفاده می کنیم. بنابراین HQL یک زبان مستقل از پایگاه داده می باشد .

مزایای استفاده از HQL

استفاده از HQL یا Hibernate Query Language مزایای زیادی دارد که از آن جمله می توان به موارد زیر اشاره کرد:

- مستقل از پایگاه داده یا .data base independent
- پشتیبانی از Polymorphic Quercy یا جستجوی چندگانه.
- امکان یادگیری ساده و آسان برای برنامه نویسان جاوا.

رابط کاربری Query Interface

Query Interfaceیک نسخه شی گرا از زبان Hibernate Query است. شی یا object متعلق به Query را می توان با فراخوانی متد ()Create Query در Session Interface ایجاد کرد.

Query Interfaceمتدهای زیادی را برای کار با داده فراهم کرده است. لیست زیر پرکاربردترین آن ها را نام می برد:

- متد : ()public int execute Update این متد برای Update یا Delate یک Query به کار می رود.
- متد : ()Public List list این متد نتایج حاصل از Query یا رابطه را به صورت list بر می گرداند.
- متد : ()public Query setMax Result این متد تعداد رکوردهایی که بایستی از رابطه (جدول پایگاه داده) استخراج شود را تعیین می کند.

- متد : public Query First Result (int rowno) این متد، شماره رکوردی که بایستی اطلاعات رابطه یا جدول پایگاه داده از آنجا به بعد استخراج شود را تعیین می کند.
- متد : public Query set Parameter (int position, Object value) این متد مقدار value را در JDBC style Queryتعیین می کند.
- متد : public Queryset Parameter (String name, Object value) این متد مقدار value را به یک. named Query Parameter ارسال می کند

کد مثال دریافت اطلاعات کلید رکوردها با استفاده از HQL

```
Query query=session.createQuery("from Emp");//here persistent class name is
Emp
List list=query.list();
```

کد مثال دریافت اطلاعات کلیه رکوردها با استفاده از HQL و صفحه بندی نتایج:

```
Query query=session.createQuery("from Emp");
query.setFirstResult(5);
query.setMaxResult(10);
List list=query.list();
//will return the records from 5 to 10th number
```

کد مثال HQL Update Query

```
Transaction tx=session.beginTransaction();
Query q=session.createQuery("update User set name=:n where id=:i");
q.setParameter("n","Udit Kumar");
q.setParameter("i",111);

int status=q.executeUpdate();
System.out.println(status);
tx.commit();
```

کد مثال HQL delete Query

```
Query query=session.createQuery("delete from Emp where id=100");
//specifying class name (Emp) not tablename
query.executeUpdate();
```

استفاده از توابع ریاضی در HQL

شما می توانید از توابع محاسباتی و ریاضی در HQL استفاده کنید. در ادامه مثال هایی برای استفاده از این توابع ارائه شده است:

کد مثال محاسبه حقوق کلیه کارمندان

```
Query q=session.createQuery("select sum(salary) from Emp");
List<emp> list=q.list();
    Iterator<emp> itr=list.iterator();
    while(itr.hasNext()){
        System.out.println(itr.next());
}
            </emp></emp>
```

کد مثال یافتن بالاترین میزان حقوق در کارمندان

```
Query q=session.createQuery("select max(salary) from Emp");
```

کد مثال یافتن کمترین میزان حقوق در کارمندان

```
Query q=session.createQuery("select min(salary) from Emp");
```

کد مثال محاسبه تعداد کارمندان درHQL

```
Query q=session.createQuery("select count(id) from Emp");
```

کد مثال یافتن میانگین حقوق کارمندان در HQL

```
Query q=session.createQuery("select avg(salary) from Emp");
```

آموزش زبان HCQL در Hibernate

زبان HCQL که مخفف عبارت Hibernate Criteria Query Language است، برای استخراج و دریافت رکوردها از یک رابطه یا جدول بر مبنای یک عبارت جستجو یا الگو خاص (Criteria) به کار می رود. رابط کاربری Criteriaیا Criteria Interface متدهایی را فراهم می کند تا به وسیله آن ها معیار مورد نظر جستجو را به

Queryخود اعمال کنیم. برای مثال اطلاعات رکوردهایی را که میزان حقوق) فیلد Salary آن ها بیشتر از 50000 است را بیابد و.(...

مزایای استفاده از HQL

زبان HCQL متدهایی را در اختیار ما قرار می دهد تا Criteria را به Query خود اضافه کنیم، بنابراین به کار بردن آن برای برنامه نویسان جاوا بسیار راحت است.

رابط کاربری Criteria Interface

رابط کاربری Criteria Interface متدهای زیادی برای تعیین Criteriaفراهم کرده است. شی یا object مربوط به Criteria را می توان با استفاده از متد ()CreateCriteria ایجاد نمود.

ساختار دستوری برای استفاده از متد createCriteria رابط کاربری Session Interface به صورت زیر است:

```
public Criteria createCriteria(Class c)
```

متدهای پرکاربرد مربوط به رابط کاربری Criteria Interface را به صورت زیر معرفی کرده ایم:

- متد : public Criteria add (Criteria c) برای اضافه کردن محدودیت های مورد نظر به Criteria استفاده می شود .
- متد : public Criteria addOrder (Order o) از این متد برای تعیین ترتیب یا order در Criteria استفاده می شود .
- متد : public Criteria setFirstResult (int first Result) این متد اولین رکوردی که بایستی از جدول یا رابطه پایگاه داده استخراج شود را تعیین می کند .
- متد : public Criteria setMaxResult (int total Result) این متد تعداد کل رکوردهایی که بایستی استخراج شوند را تعیین می کند .
- متد : public List list() این متد یک listحاویobject های برنامه را بر می گرداند .
- متد : public Criteria setProjection (Projection projection) این متد projection را در کد تعیین می کند.

آموزش کلاس های محدودکننده Restriction Class در HQL

کلاس های محدودکننده یا Restriction Class ها، متدهایی را فراهم می کنند که از آن ها می توان برای ایجاد معیار (Criterion) استفاده کرد. متدهای پرکاربردی که می توان آن ها را به عنوان Restriction به کار برد، عبارتند از:

- متد : public static SimpleExpression lt (String propertyName,Object value) این متد مقدار کمتر از محدودیت less than Contrasnt را به property مورد نظر می دهد.
- متد : public static SimpleExpression le (String propertyName,Object value) مقدار کمتر یا مساوی محدودیت less than or equal را به property مورد نظر می دهد.
- متد : public static SimpleExpression gt (String propertyName,Object value) مقداربیشتر از محدودیت grater than را به property مورد نظر می دهد.
- متد : public static SimpleExpression ge (String propertyName,Object value) مقدار بیشتر یا مساوی محدودیت grater than or equal را به property مورد نظر می دهد.
- متد : public static SimpleExpression ne (String propertyName,Object value) مقدار نابرابر و غیر همسان با محدودیت not equal را به property مورد نظر می دهد.
- متد : public static SimpleExpression eq (String propertyName,Object value) مقدار مساوی و برابر با محدودیت تعیین شده را به property مورد نظر می دهد.
- متد public static Criterion between(String propertyName, Object low, Object high) :این متد محدوده مجاز مقدار محدودیت را تعیین می کند.
- متد : public static SimpleExpression like(String propertyName, Object value) این متد مقدار شبیه مقدار محدودیت like constraint را به property مورد نظر می دهد.

مثال کد نوشتن با زبان HCQL در Hibernate

کد مثال خواندن تمامی رکوردها با استفاده از HCQL

```
Crietria c=session.createCriteria(Emp.class);//passing Class class argument
List list=c.list();
```

کد مثال خواندن رکوردهای 10 ام تا 20 ام در HCQL

```
Crietria c=session.createCriteria(Emp.class);
c.setFirstResult(10);
c.setMaxResult(20);
List list=c.list();
```

کد مثال خواندن رکوردهایی که میزان salary آن ها بیشتر از 10000 است به وسیله HCQL

```
Crietria c=session.createCriteria(Emp.class);
c.add(Restrictions.gt("salary",10000));//salary is the propertyname
List list=c.list();
```

آموزش HCQL با Projection

```
Crietria c=session.createCriteria(Emp.class);
c.addOrder(Order.asc("salary"));
List list=c.list();
```

شما می توانید با استفاده از Projection اطلاعات مربوط به یک فیلد یا Columnخاص مثل ستون name را استخراج کنید. در کد مثال زیر، برنامه فقط اطلاعات مربوط به ستون NAME از جدول مورد نظر را چاپ می کند :

```
Criteria c=session.createCriteria(Emp.class);
c.setProjection(Projections.property("name"));
List list=c.list();
```

آموزش Hibernate Named Query

Hibernate Named Querراهی است برای استفاده از Query در سطح برنامه با استفاده از نام های معنی دار. به عبارت دیگر یک نام هم معنی با کارکرد Query برای آن انتخاب کرده و به وسیله آن نام Query مورد نظر را در برنامه های Hibernate فراخوانی و اجرا می کنیم.

Hibernate Named Querهمانند کاربرد alias در زبان SQL است. چهارچوب کاری Hibernate امکان استفاده از Named Query ها را فراهم کرده و به همین دلیل دیگر برنامه نویسان جاوا نیاز ندارند تا Queryرا در کل سطح برنامه پراکنده کنند.

دو روش برای تعیین Named Query در Hibernate وجود دارد:

- استفاده از annotation.
- استفاده از فایل mapping file.

آموزش Hibernate Named Query با استفاده از annotation

اگر می خواهید از قابلیت Named Query در Hibernate استفاده کنید، بایستی با نحوه کار با annotation های @NamedQueries و @NamedQuery آشنا باشید.

@NamedQueries annotation برای تعیین Query های چند نامه به کار می رود.

@NamedQuery برای تعیین Query تک نام به کار می رود.

در کد زیر، مثال عملی استفاده از Named Query در Hibernate نشان داده شده است:

```
@NamedQueries(
    {
        @NamedQuery(
        name = "findEmployeeByName",
        query = "from Employee e where e.name = :name"
        )
    }
)
```

مثال عملی آموزش Hibernate Named Query با استفاده از annotation

در این مثال، از annotation برای تعیین Named Query در کلاس Persistant Classاستفاده می کنیم.

این مثال دارای 3 فایل اصلی به شرح زیر است:

- Employee.java.

- hibernate.cfg.xml.
- FetchDemo.

فایل Employee.java

فایل زیر کد یک کلاس Persistent Classs است که از annotationبرای تعیین Named Query استفاده کرده و آن را به عنوان یک موجودیت یا entity می گیرد.

```
package com.javatpoint;

import javax.persistence.*;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;

@NamedQueries(
    {
        @NamedQuery(
        name = "findEmployeeByName",
        query = "from Employee e where e.name = :name"
        )
    }
)

@Entity
@Table(name="em")
public class Employee {

    public String toString(){return id+" "+name+" "+salary+" "+job;}

    int id;
    String name;
    int salary;
    String job;
    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)

    //getters and setters
}
```

فایل hibernate.cfg.xml

فایل زیر یک فایل تنظیمات یا Configuration file است که اطلاعات مهم برنامه از قبیلdriver class ، Url، User name، password، mapping class وغیره را نگهداری می کند.

```
<!--?xml version='1.0' encoding='UTF-8'?-->


<hibernate-configuration>

    <session-factory>
        <property name="hbm2ddl.auto">update</property>
        <property
name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
        <property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
        <property name="connection.username">system</property>
        <property name="connection.password">oracle</property>
        <property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
    <mapping class="com.javatpoint.Employee">
    </mapping></session-factory>

</hibernate-configuration>
```

فایل FetchData.java

کد زیر کلاس FetchData است که از Named Query تعیین شده استفاده کرده و اطلاعات مورد نظر را بر مبنای Query از پایگاه داده استخراج می کند. متد getNamedQuery از Named Query استفاده کرده و یک نسخه یا instance از Query را بر می گرداند.

```
package com.javatpoint;

import java.util.Iterator;
import java.util.List;

import org.hibernate.cfg.AnnotationConfiguration;
import org.hibernate.*;

public class FetchData {
 public static void main(String[] args) {

  AnnotationConfiguration configuration=new AnnotationConfiguration();
   configuration.configure("hibernate.cfg.xml");
   SessionFactory sFactory=configuration.buildSessionFactory();
    Session session=sFactory.openSession();

    //Hibernate Named Query
    Query query = session.getNamedQuery("findEmployeeByName");
    query.setString("name", "amit");

    List< Employee > employees=query.list();

    Iterator< Employee > itr=employees.iterator();
     while(itr.hasNext()){
    Employee e=itr.next();
    System.out.println(e);
     }
```

```
    session.close();

  }
}
```

مثال عملی آموزش Named Query با استفاده از mapping file

اگر بخواهید Named Query را با استفاده از mapping file تعیین کنید، بایستی از المنت query فایل hibernate-mappingبرای این کار استفاده کنید.

در چنین حالتی، شما بایستی hbm فایل مورد نظر جهت تعیین Named Queryرا ایجاد نمایید. فایل های دیگر این مثال به جز فایل کلاس Persistent Classبه نام Employee.java که در آن نیازی نیست برای تعیین Named Queryاز هیچ annotation ای استفاده کرده و فایل تنظیمات به نام hibernate.cfg.xml که در آن بایستی منبع لازم جهت mapping را در فایل hbm تعین کنید، همانند مثال قبل هستند.

فایل hbm مثال Named Query با استفاده از mapping file بایستی به صورت زیر باشد:

فایل emp.hbm.xml

```
<!--?xml version='1.0' encoding='UTF-8'?-->


<hibernate-mapping>
<class name="com.javatpoint.Employee" table="em">
<id name="id">
<generator class="native"></generator>
</id>
<property name="name"></property>
<property name="job"></property>
<property name="salary"></property>
</class>

<query name="findEmployeeByName">
<!--[CDATA[from Employee e where e.name = :name]]-->
</query>

</hibernate-mapping>
```

همچنین فایل کلاس Persistent Class نیز بایستی همانند زیر باشد :

```
package com.javatpoint;
public class Employee {
    int id;
```

```
    String name;
    int salary;
    String job;
    //getters and setters
}
```

سپس در مرحله آخر، منبع لازم جهت mapping یعنی mapping resource را در فایل hbm به صورت کد زیر اضافه کنید:

```
<mapping resource="emp.hbm.xml">
</mapping>
```

آموزش قابلیت Caching در Hibernate

قابلیت Caching Hibernate با قرار دادن اشیا object در حافظه، باعث بالارفتن کارایی برنامه می شود .

دو مدل کلی برای Caching در Hibernate وجود دارد:

- first level Cache.
- second level Cache.

آموزش First Level Cache

شی Session Object دیتای first level cache را نگهداری می کند. این شی به صورت پیش فرض فعال است. دیتای first level cache برای کل برنامه در دسترس نیست. هر Hibernate Application می توانند تعداد زیادی session object استفاده کند.

آموزش Second level Cache

شی SessionFactory Object دیتای second level cache را نگهداری می کند. دیتایی که در cache second level نگهداری می شود برای کل برنامه قابل دسترس می باشد. اما بایستی آن را به صورت دستی فعال کنید.

آموزش Second Level Cache در Hibernate

- کش Second Level Cache Hibernate از یک کش اشتراکی Common Cache برای تمامی Session objectهای شی Session factory استفاده می کند .این کش در صورتی که چندین شی Session objectاز یک Session factory داشته باشید، بسیار کارآمد است. شی Session Factoryاطلاعات second level را نگهداری می کند. این شی برای کلیه session object ها عمومی بوده و به صورت پیش فرض فعال نیست.

چندین برنامه مختلف می توانند قابلیت Second Level Cache را در برنامه فعال کنند:

- EH Cache
- OS Cache
- Swarm Cache
- JBass Cache

هر یک از اجراهای فوق(implementation) ، کارکردهای مختلفی از قابلیت Cache را فراهم می کند. چهار راه برای استفاده از Second level Cache وجود دارد که عبارتند از:

- **read-only** :قابلیت Caching فقط برای عملیات خواندنی read-only فعال است.
- **nonstrict-read-write** :قابلیت Cachingهم برای عملیات خواندن (read) و هم عملیات نوشتن (write)به کار می رود ولی در هر زمان فقط از یکی از عملیات ها می توان استفاده کرد.
- **read-write** :قابلیت Caching به صورت هم زمان هم برای عملیات خواندن (read) و هم برای عملیات نوشتن (write)به کار می رود.
- **transactional** :قابلیت Caching برای عملیات transaction فعال می شود.

خاصیت Cache-usage را می توان به هر کلاس class یا Collection level در فایل hbm.xml اضافه کرد.

به صورت کد زیر:

```
<cache usage="read-only">
            </cache>
```

جدول زیر نیز نحوه اجرای Second Level Cache و موارد استفاده Cache usage را نشان می دهد:

Implementation	read-only	nonstrict-read-write	read-write	transactional
EH Cache	Yes	Yes	Yes	No
OS Cache	Yes	Yes	Yes	No
Swarm Cache	Yes	Yes	No	No
JBoss Cache	No	No	No	Yes

3مرحله اضافی که بایستی برای اجرای Second level cache به وسیله EH cacheانجام دهید، عبارتند از:

1. 2تنظیم زیر را در فایل Configuration برنامه، فایل hibernate.cfg.xml اضافه کنید.

```
<property
name="cache.provider_class">org.hibernate.cache.EhCacheProvider</property>
<property name="hibernate.cache.use_second_level_cache">true</property>
```

2. تنظیمات مربوط به Cache usage را در فایل hbm اضافه کنید:

```
<cache usage="read-only">
            </cache>
```

3. فایل ehcache.xml را به صورت زیر ایجاد و به پروژه اضافه کنید:

```
<!--?xml version="1.0"?-->
<ehcache>

<defaultcache maxelementsinmemory="100" eternal="true">

</defaultcache></ehcache>
```

مثال عملی آموزش Second Level Cache درHibernate

برای اجرا و آموزش مثال second level cache در Hibernate ابتدا بایستی فایل های زیر را ایجاد و به پروژه اضافه کنید:

کد فایل : Employee.java

```
package com.javatpoint;

public class Employee {
private int id;
private String name;
private float salary;

public Employee() {}
public Employee(String name, float salary) {
    super();
    this.name = name;
    this.salary = salary;
}
//setters and getters
}
```

کد فایل : employee.hbm.xml

```
<!--?xml version='1.0' encoding='UTF-8'?-->


          <hibernate-mapping>
          <class name="com.javatpoint.Employee" table="emp1012">
          <cache usage="read-only">
          <id name="id">
          <generator class="native"></generator>
          </id>
          <property name="name"></property>
          <property name="salary"></property>
          </cache></class>

</hibernate-mapping>
```

در کد مثال فوق، ما از مدل read-only برای Cache only استفاده کرده ایم Cache usage. را نیز می توان در Collection استفاده کرد.

فایل : hibernate.cfg.xml

```
<!--?xml version='1.0' encoding='UTF-8'?-->


<!-- Generated by MyEclipse Hibernate Tools.                    -->
<hibernate-configuration>

    <session-factory>
        <property name="show_sql">true</property>
        <property name="hbm2ddl.auto">update</property>
        <property
name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
```

```
        <property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
        <property name="connection.username">system</property>
        <property name="connection.password">oracle</property>
        <property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>

        <property
name="cache.provider_class">org.hibernate.cache.EhCacheProvider</property>
         <property
name="hibernate.cache.use_second_level_cache">true</property>

    <mapping resource="employee.hbm.xml">
    </mapping></session-factory>

</hibernate-configuration>
```

برای اجرای second level cache ، بایستی خاصیت Cache.provider_class را در فایل تنظیمات Configuration fileاضافه کنیم.

فایل ehcache.xml

```
<!--?xml version="1.0"?-->
<ehcache>
<defaultcache maxelementsinmemory="100" eternal="false"
timetoidleseconds="120" timetoliveseconds="200">

<cache name="com.javatpoint.Employee" maxelementsinmemory="100"
eternal="false" timetoidleseconds="5" timetoliveseconds="200">
</cache></defaultcache></ehcache>
```

برای تعیین خاصیت Property بایستی فایل ehcache.xml را ایجاد کنید.

مقدار defaultCache برای کلیه کلاس های Persistent Class برنامه به کار می رود. شما می توانید Persistent Classرا به صورت مستقیم توسط المنت Cache element نیز تعیین کنید.

در مورد خاصیت eternal باید اشاره کرد اگر مقدار خاصیت "eternal="true قرار گیرد، دیگر نیازی نیست خواص timeToLiveSeconds و timeToIdleSecondsرا تعیین کنیم زیرا به صورت اتوماتیک توسط hibernateمدیریت می شود.

تعیین خاصیت "eternal="false کنترل کامل برنامه را به دست برنامه نویس داده ولی بایستی خواص TimetoLiveSecondsو timeToIdleSeconds را به صورت دستی تعیین کنیم.

خاصیت timeToIdleSeconds تعیین می کند هر شی یا object چند ثانیه می تواند در Cache به صورت Idleباشد.

همچنین خاصیت timeToLiveSeconds تعیین می کند که هر شی یا objectچند ثانیه می تواند در حافظه Cacheچه به صورت Idle یا Live حضور داشته باشد.

فایل FetchTest.java

```
package com.javatpoint;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class FetchTest {
public static void main(String[] args) {
Configuration cfg=new Configuration().configure("hibernate.cfg.xml");
SessionFactory factory=cfg.buildSessionFactory();

    Session session1=factory.openSession();
    Employee emp1=(Employee)session1.load(Employee.class,121);
    System.out.println(emp1.getId()+" "+emp1.getName()+" "+emp1.getSalary());
    session1.close();

    Session session2=factory.openSession();
    Employee emp2=(Employee)session2.load(Employee.class,121);
    System.out.println(emp2.getId()+" "+emp2.getName()+" "+emp2.getSalary());
    session2.close();

}
}
```

خروجی:

خروجی برنامه در نهایت به صورت زیر است:

```
Console
<terminated> FetchTest [Java Application] C:\Users\Sonoo Jaiswal\AppData\Local\Genuitec\Common\bina
log4j:WARN No appenders could be found for logger (org.hibernate.cfg.En
log4j:WARN Please initialize the log4j system properly.
Hibernate: select employee0_.id as id0_0_, employee0_.name as name0_0_,
121 Rahul 50000.0
121 Rahul 50000.0
```

همان طور که در خروجی فوق مشاهده می کنید، hibernateهر queryرا دو بار اجرا یا fire نکرده است. اگر از Second Level Cacheاستفاده نکنید، به دلیل این که هر query از session object های متفاوتی استفاده می کند، queryبرنامه دو بار اجرا می شود.

آموزش ادغام برنامه های Struts و Hibernate

شما می توانید هر برنامه struts2 را با برنامه های Hibernateبه راحتی ادغام کنید. برای انجام این کار تلاش اضافه ای لازم نیست.

در این مثال آموزشی قصد داریم تا چهارچوب کاری Struts2 را با Hibernate به کار ببریم. برای این منظور نیاز دارید فایل های jar لازم جهت Struts2 و Hibernate را داشته باشید.

مثال آموزش ادغام برنامه های Struts 2 و Hibernate

در مثال آموزشی این درس، قصد داریم تا به وسیله struts2 یک فرم ثبت registration form را ایجاد کرده و سپس اطلاعات وارد شده در فرم را به وسیله Hibernate در پایگاه داده ذخیره کنیم.

در لیست زیر فایل هایی که لازم هستند برای ادغام struts2 و Hibernate ایجاد کنیم را معرفی کرده ایم:

- فایل index.jsp که داده ورودی یا input را از کاربر دریافت می کند.
- فایل User.java که حاوی یک کلاس عملی action class برای مدیریت درخواست کاربر می باشد. این فایل از کلاس dao برای ذخیره اطلاعات استفاده می کند.
- فایل RegisterDao.java که شامل یک کلاس java بوده و از قالب طراحی DAO برای نگهداری اطلاعات به وسیله Hibernate استفاده می کند.
- فایل User.hbm.xml یک فایل آدرس دهی mapping file بوده و حاوی اطلاعات لازم درباره کلاس Persistent Classمی باشد. در این مثال، کلاس عملی Action Class به عنوان کلاس Persistent Classایفای نقش خواهد کرد.
- فایل تنظیمات hibernate.cfg.xml که یک فایل configuration fileبوده و حاوی اطلاعات لازم درباره پایگاه داده و فایل آدرس دهی mapping file است.

- فایل struts.xml که شامل اطلاعات لازم درباره action class و صفحه نتایج result Page بوده تا فراخوانی شود.
- فایل Welcom.jsp یک فایل jsp بوده و اطلاعات لازم جهت خوش آمد گویی به کاربر و نام کاربری Usernameآن را نشان می دهد.
- در نهایت نیز فایل web.xml که یک فایل xml بوده و حاوی اطلاعات لازم جهت Controller مورد نظر برای چهارچوب کاری struts می باشد.

در ادامه کد هر یک از فایل های فوق را بررسی می کنیم.

فایل : index.java

در این صفحه، یک form را به وسیله تگ های struts ایجاد کرده ایم Action name. این form مقدار registerاست.

```
<%@ taglib uri="/struts-tags" prefix="S" %>

<s:form action="register">
<s:textfield name="name" label="Name"></s:textfield>
<s:submit value="register"></s:submit>

</s:form>
```

فایل : User.java

این فایل یک کلاس ساده PoJo است. در این مثال، فایل User.javaبه عنوان action class برای struts و Persistent classبرای Hibernateکار می کند. این کلاس متد registerکلاس Register Dao را فراخوانی کرده و نتیجه را به صورت متنی یا یک string باز می گرداند.

```
package com.javatpoint;

public class User {
private int id;
private String name;
//getters and setters

public String execute(){
    RegisterDao.saveUser(this);
    return "success";
```

```
}

}
```

فایل : RegisterDao.java

فایل RegisterDao.java یک کلاس java می باشد که object مربوط به UserClass را به وسیله Hibernate ذخیره می کند.

```
package com.javatpoint;

import org.hibernate.Session;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;

public class RegisterDao {

public static int saveUser(User u){

Session session=new Configuration().
configure("hibernate.cfg.xml").buildSessionFactory().openSession();

Transaction t=session.beginTransaction();
int i=(Integer)session.save(u);
t.commit();
session.close();

return i;

}

}
```

فایل : user.hbm.xml

این فایل آدرس دهی mapping file شامل تمامی اطلاعات مربوط به کلاس Persistent Class می شود.

```
<!--?xml version='1.0' encoding='UTF-8'?-->


<hibernate-mapping>
<class name="com.javatpoint.User" table="user451">
<id name="id">
<generator class="increment"></generator>
</id>
<property name="name"></property>
</class>
```

```
</hibernate-mapping>
```

فایل : hibernate.cfg.xml

این فایل تنظیمات Configuration file شامل کلیه اطلاعات مربوط به پایگاه داده database و فایل آدرس دهی mapping file می شود.

در این مثال ما از خاصیت hb2ddl.auto استفاده کرده ایم، بنابراین دیگر نیاز ندارید تا جدول در پایگاه داده درست کنید.

```
<!--?xml version='1.0' encoding='UTF-8'?-->


<!-- Generated by MyEclipse Hibernate Tools.                    -->
<hibernate-configuration>

<session-factory>
<property name="hbm2ddl.auto">update</property>
<property name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
<property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
<property name="connection.username">system</property>
<property name="connection.password">oracle</property>
<property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
<mapping resource="user.hbm.xml">

</mapping></session-factory>

</hibernate-configuration>
```

فایل : Struts.xml

فایل struts.xml شامل اطلاعات لازم جهت فراخوانی action class می باشد. در این فایل action class همان User است.

```
<!--?xml version="1.0" encoding="UTF-8" ?-->

<struts>

<package name="abc" extends="struts-default">
<action name="register" class="com.javatpoint.User">
<result name="success">welcome.jsp</result>
</action>
</package>
```

```
</struts>
```

فایل : Welcome.jsp

فایل welcome.jsp پیام خوش آمد گویی به همراه نام کاربری username را نمایش می دهد.

```
<%@ taglib uri="/struts-tags" prefix="S" %>

Welcome: <s:property value="name">
            </s:property>
```

فایل : Web.xml

فایل web.xml شامل اطلاعات لازم درباره Controller است. در چهارچوب کاری Struts2 ، کلاس StrutsPropare And Excute Filter به عنوان کنترلر عمل می کند.

```
<!--?xml version="1.0" encoding="UTF-8"?-->
<web-app version="2.5" xmlns="java.sun.com/xml/ns/javaee"
xmlns:xsi="www.w3.org/2001/XMLSchema-instance"
xsi:schemalocation="java.sun.com/xml/ns/javaee
    java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
  <filter>
    <filter-name>struts2</filter-name>
    <filter-class>
  org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter
    </filter-class>
  </filter>
  <filter-mapping>
    <filter-name>struts2</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
</web-app>
```

آموزش ادغام برنامه های Spring و Hibernate

شما به سادگی می توانید برنامه های Spring و Hibernate را با هم ادغام کنید.

در چهارچوب کاری Hibernate کلیه اطلاعات لازم جهت پایگاه داده را در فایل Hibernate.cfg.xml تعیین می

کنیم.

اما اگر بخواهیم یک برنامه Hibernate را با یک برنامه Springادغام کنیم، دیگر نیازی به ایجاد فایل hibernate.cfg.xmlنیست. می توانیم کلیه اطلاعات مورد نیاز را در فایل applicationContext.xml قرار بدهیم.

مزایای استفاده چهارچوب کاری Spring با Hibernate

چهارچوب کاریSpring ، کلاس HibernateTemplate را فراهم کرده است. به همین دلیل دیگر نیاز نیست مراحل مختلفی مثل ایجاد فایل تنظیماتConfiguration file ، ساخت فایل BulidSessionFactory ، ایجاد session، انجام شروع و پایان تراکنش ها begin and commit transaction را انجام دهیم.

بنابراین استفاده چهارچوب کاری Spring در Hibernate به طور قابل ملاحظه ای حجم کد برنامه را کاهش می دهد.

درک مشکل عدم استفاده از Spring با Hibernate

بیایید این مسئله را با بررسی کد فایل hibernate بهتر درک کنیم:

```
//creating configuration
Configuration cfg=new Configuration();
cfg.configure("hibernate.cfg.xml");

//creating seession factory object
SessionFactory factory=cfg.buildSessionFactory();

//creating session object
Session session=factory.openSession();

//creating transaction object
Transaction t=session.beginTransaction();

Employee e1=new Employee(111,"arun",40000);
session.persist(e1);//persisting the object

t.commit();//transaction is commited
session.close();
```

همان طور که در کد فایل Hibernate فوق مشاهده می کنیم، در Hibernate بایستی مراحل زیادی را انجام دهید.

اما راه حل، استفاده از کلاس HibernateTemplate به وسیله چهارچوب کاری Spring است.

در این حالت، نیاز نیست مراحل زیادی را طی کنید. فقط کافی است کد زیر را بنویسید:

```
Employee e1=new Employee(111,"arun",40000);
hibernateTemplate.save(e1);
```

آموزش متدهای پرکاربرد کلاس HibernateTemplate

1. متد Void Persistent (object entity) : این متد objectداده شده به ان را در برنامه نگهداری می کند.
2. متد Serializable save (object entity) : این متد object داده شده به آن را نگهداری کرده و id شی را بر می گرداند.
3. متد Void save OrUpdate(object entity) : این متد object داده شده را نگهداری یا به روزرسانی Update می کند. اگر id شی پیدا شد، متد رکورد آن را به روز یا Update می کند، در غیر این صورت رکورد را ذخیره می کند.
4. متد Void update (object entity) : این متد شی داده شده را به روز یا Update می کند.
5. متد Void delete(object entity) : این متد object داده شده را بر حسب id آن حذف یا delete می کند.
6. متد Object get (Class entity Class, Ser ializable id) : این متد شی Persistent object را بر حسب id داده شده بر می گرداند.
7. متد Object load (Class entityClass,Serializable id) : این متد شی Persistent object را بر حسب id داده شده، لود یا اجرا می کند.
8. متد List loadAll (Class entityclass) : این متد کلیه شی های Persistent Object را بر می گرداند.

مراحل کار:

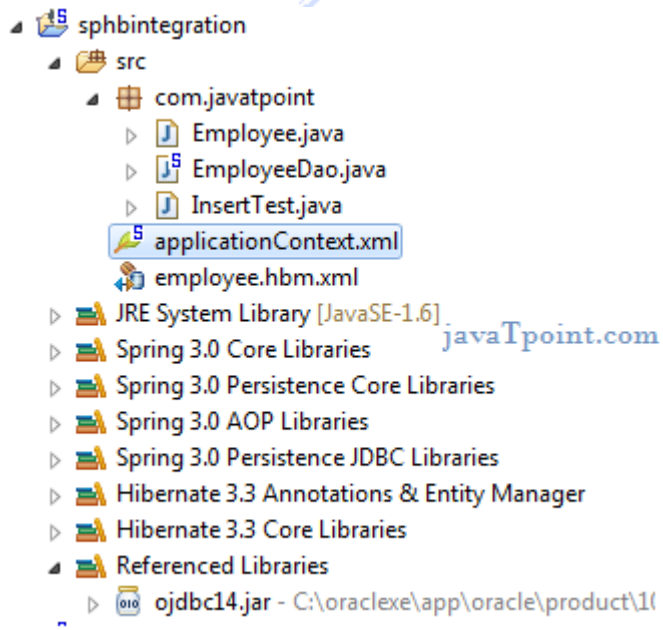
در این بخش مراحل مختلف نحوه ادغام برنامه های Hibernate و Spring را نشان داده ایم:

1. ایجاد جدول در پایگاه داده، که این مرحله اختیاری است.
2. ایجاد فایلapplicationContext.xml ، که این فایل دربرگیرنده اطلاعات لازم دربارهDataSource ، SessionFactoryو ... می باشد.
3. ایجاد فایل Employee.java که این فایل کلاس Persistent Class را تعیین می کند.
4. ایجاد فایل employee.xml که این فایل نیز فایل آدرس دهی mapping file را تعیین می کند.
5. ایجاد فایلEmployeeDao.java ، که این فایل کلاس dao class ای که از Hibernate Templateاستفاده می کند را تعیین می کند.
6. ایجاد فایل InsertTest.java که این فایل متدهای لازم کلاس EmployeeDao Class را فراخوانی می کند.

مثال عملی آموزش ادغام برنامه های Spring و Hibernate

در این مثال عملی، قصد داریم تا روش ادغام برنامه های Hibernate و Spring را آموزش دهیم.

تصویر زیر، کلیه فایل ها و پوشه های مثال Spring و Hibernate را نشان می دهد:



1. ایجاد جدول در پایگاه داده:

در این مثال ما از Oracle برای ایجاد پایگاه داده استفاده کرده ایم، اما شما می توانید از سایر پایگاه های داده دیگر نیز استفاده کنید. به وسیله کد زیر، جدول مورد نظر خود را در پایگاه داده ایجاد می کنیم :

```
CREATE TABLE  "EMP558"
   (    "ID" NUMBER(10,0) NOT NULL ENABLE,
    "NAME" VARCHAR2(255 CHAR),
    "SALARY" FLOAT(126),
     PRIMARY KEY ("ID") ENABLE
   )
/
```

2. فایل: Employee.java

این فایل یک کلاس ساده PoJo Class است که به عنوان کلاس Persistent Class برای hibernateکار می کند:

```
package com.javatpoint;

public class Employee {
private int id;
private String name;
private float salary;

//getters and setters

}
```

3. فایل: employee.hbm.xml

این فایل یک فایل آدرس دهی mapping file بوده که حاوی یک اطلاعات لازم جهت کلاس Persistent Classاست:

```
<!--?xml version='1.0' encoding='UTF-8'?-->


<hibernate-mapping>
<class name="com.javatpoint.Employee" table="emp558">
          <id name="id">
          <generator class="assigned"></generator>
          </id>
```

```
        <property name="name"></property>
        <property name="salary"></property>
</class>

</hibernate-mapping>
```

4. فایل: EmployeeDao.java

فایل EmployeeDao.java یک کلاس java class بوده که از متدهای کلاس Hibernate Template برای ایجاد شی یا object کلاس Employee Class استفاده می کند:

```
package com.javatpoint;
import org.springframework.orm.hibernate3.HibernateTemplate;
import java.util.*;
public class EmployeeDao {
HibernateTemplate template;
public void setTemplate(HibernateTemplate template) {
    this.template = template;
}
//method to save employee
public void saveEmployee(Employee e){
    template.save(e);
}
//method to update employee
public void updateEmployee(Employee e){
    template.update(e);
}
//method to delete employee
public void deleteEmployee(Employee e){
    template.delete(e);
}
//method to return one employee of given id
public Employee getById(int id){
    Employee e=(Employee)template.get(Employee.class,id);
    return e;
}
//method to return all employees
public List<employee> getEmployees(){
    List<employee> list=new ArrayList<employee>();
    list=template.loadAll(Employee.class);
    return list;
}
}
            </employee></employee></employee>
```

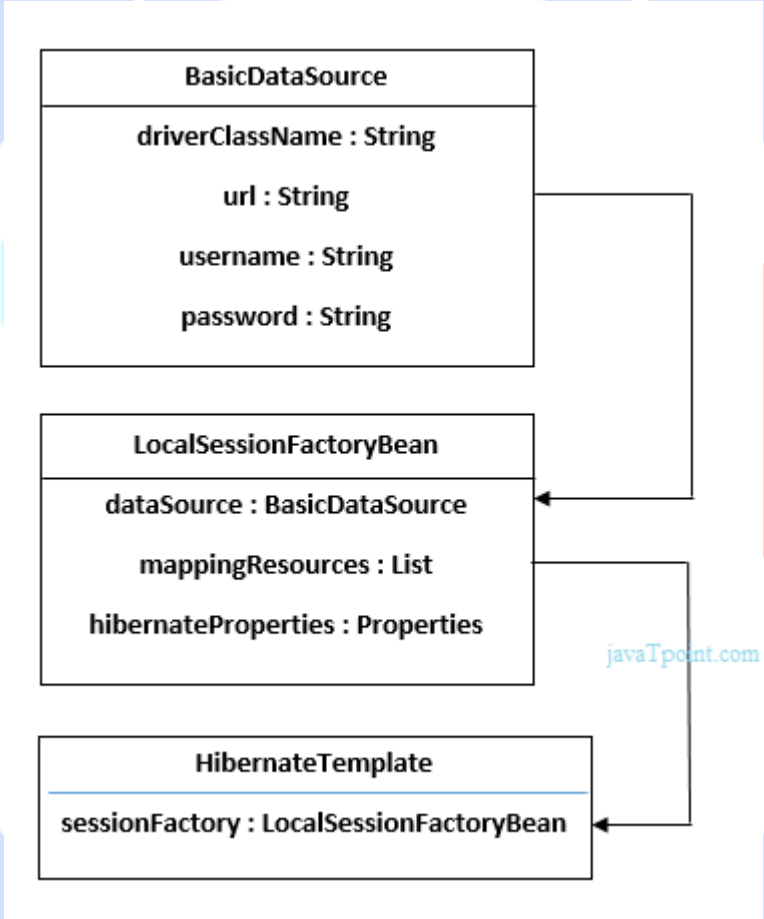
5. فایل: application Context.xml

در فایل applicationContext.xml ، کلیه اطلاعات لازم جهت پایگاه داده را در شی

BasicDataSource Objectفراهم کرده ایم. این فایل در شی کلاس LocalSession Factory Beanاستفاده شده و حاوی اطلاعات دیگری از جمله mappingResource و hibernatePropertiesمی باشد.

شی LocalSession Factory Bean در کلاس Hibernate Template Class استفاده می شود.

شکل زیر ساختار کد لازم جهت ایجاد فایل applicationContext.xmlرا نشان می دهد:




```
<!--?xml version="1.0" encoding="UTF-8"?-->
<beans xmlns="www.springframework.org/schema/beans"
xmlns:xsi="www.w3.org/2001/XMLSchema-instance"
xmlns:p="www.springframework.org/schema/p"
xsi:schemalocation="www.springframework.org/schema/beans
        www.springframework.org/schema/beans/spring-beans-3.0.xsd">


    <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
        <property name="driverClassName"
value="oracle.jdbc.driver.OracleDriver"></property>
        <property name="url"
value="jdbc:oracle:thin:@localhost:1521:xe"></property>
```

```
        <property name="username" value="system"></property>
        <property name="password" value="oracle"></property>
    </bean>

    <bean id="mysessionFactory"
class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
        <property name="dataSource" ref="dataSource"></property>

        <property name="mappingResources">
        <list>
        <value>employee.hbm.xml</value>
        </list>
        </property>

        <property name="hibernateProperties">
            <props>
                <prop
key="hibernate.dialect">org.hibernate.dialect.Oracle9Dialect</prop>
                <prop key="hibernate.hbm2ddl.auto">update</prop>
                <prop key="hibernate.show_sql">true</prop>

            </props>
        </property>
    </bean>

    <bean id="template"
class="org.springframework.orm.hibernate3.HibernateTemplate">
    <property name="sessionFactory" ref="mysessionFactory">></property>
    </bean>

    <bean id="d" class="com.javatpoint.EmployeeDao">
    <property name="template" ref="template">></property>
    </bean>


    </beans>
```

6. فایل: Insert Test.java

این کلاس از شی EmployeeDao class object استفاده کرده و با پاس دادن شی Employee Class objectمتد saveEmployee را فراخوانی می کند.

```
package com.javatpoint;

import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.ClassPathResource;
import org.springframework.core.io.Resource;

public class InsertTest {
public static void main(String[] args) {
```

```
    Resource r=new ClassPathResource("applicationContext.xml");
    BeanFactory factory=new XmlBeanFactory(r);

    EmployeeDao dao=(EmployeeDao)factory.getBean("d");

    Employee e=new Employee();
    e.setId(114);
    e.setName("varun");
    e.setSalary(50000);

    dao.saveEmployee(e);

}
}
```

اکنون اگر به جدول ایجاد شده در پایگاه داده Cracle مثال دقت کنید، خواهید دید که رکورد جدید به جدول اضافه شده است.

آموزش فعال کردن قابلیت ایجاد خودکار جدول ها (table creation) ، نشان دادنquery های SQL و: ...

شما می توانید خیلی از خواص hibernate Properties مثل ایجاد خودکار جدول (automatic table Creation)را به وسیله فایل hbm2ddl.auto در فایل applicationContext.xml فعال کنید. کد آن به صورت زیر است:

```
<property name="hibernateProperties">
            <props>
                <prop
key="hibernate.dialect">org.hibernate.dialect.Oracle9Dialect</prop>
                <prop key="hibernate.hbm2ddl.auto">update</prop>
                <prop key="hibernate.show_sql">true</prop>

</props>
            </property>
```

اگر کد فوق را به برنامه خود اضافه کنید، دیگر نیازی ندارید تا جدول پایگاه داده را به صورت دستی ایجاد نمایید. زیرا برنامه جدول را به صورت اتوماتیک ایجاد می کند.