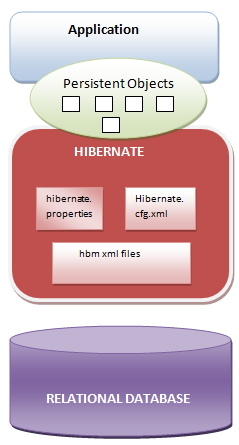
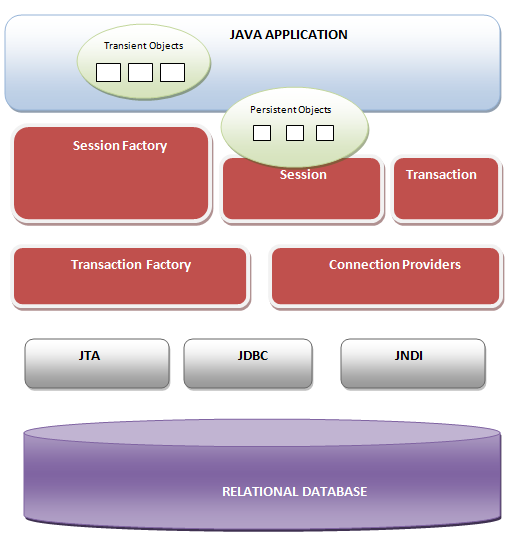
1. In relational database data are stored in the tabular format
2. Boilerplate code or boilerplate refers to sections of code that have to be included in many places with little or no alteration.
3. To map the Object to relational databases there are ORM (Object Relational Mapping) frameworks available like iBatis, Spring DAO, Hibernate.

1. Hibernate internally uses JDBC but why the hibernate is faster than the JDBC?

Ans: it is because of the following reason.

caching, optimistic locking and so on

1. Advantages:
2. Provides support for over 30 dialects
3. Hibernate provides high performance
4. Any change at a database level can be accommodated by changing the configuration file
5. Only the changed properties of an object are updated in the database instead of updating complete row
6. Hibernate provides a tool to auto generate schema from configuration files.
7. We can map multiple Java objects to one table.
8. Hibernate is one of the JPA provider which means Hibernate implements the JPA specifications.
9. For hibernate we need to provide the following information,
10. Database Information
11. Metadate of the Object like Table name, column name, data type,class name and etc
12. Configuration object is created to load the configuration files like hibernate.proerties, hibernatate.cfg.xml and hbm files (Java objects to database mapping).

**Syntax: Configuration configuration = new Configuration().configure();**

1. In case of customized file name then the syntax is,

new Configuration().configure("/configurations/myConfiguration.cfg.xml")

1. SessionFactory: is heavy weight and only one per Database,

**Syntax**: Configuration configuration = new Configuration().configure();

SessionFactory sessionFactory = configuration.buildSessionFactory();

1. Session objects are not thread safe and hence should not keep open for a long time.

Syntax:

Configuration configuration = new Configuration().configure();

SessionFactory sessionFactory = configuration.buildSessionFactory();

Session session = sessionFactory.openSession();

1. Hibernate has 3 states for the object,
2. **Persistent Object -** Persistent instances are the objects that are currently mapped with a session and has a corresponding row in a relational database.
3. **Transient Objects**- Transient instances are the objects which are never associated with a session or with a persistence context.
4. **Detached Objects-** Detached instances are the objects that are mapped to the session or persistent context but session is closed later. In other word, objects whose mapped session is closed.
5. A transaction is a unit of work in which either all operations must execute or none of them.
6. **what is the purpose of catalina in the apache tomcat server stack overflow?**

Ans: Tomcat is actually composed of a number of components, including a Tomcat JSP engine, servlet container and other connectors, but the main component is Catalina, which provides the implementation of the servlet specification. When starting the Tomcat server, it's Catalina that is actually starting. The variable 'catalina\_home' is a configuration property that stores the location of the Catalina files.

1. There are three approaches with which we can do the configurations. These approaches are-
2. **Programmatic Configurations** – Hibernate does provides a way to load and configure database and connection details programmatically.
3. **XML configurations** – We can provide the database details in an XML file. By default hibernate loads the file with name hibernate.cfg.xml but we can load the file with custom name as well.
4. **Properties configurations- This approach uses a properties file for the configuration. By default hibernate loads the file with name hibernate. properties but we can load the file with custom name as well.**
5. **Programmatic Configurations:**
6. **addClass()-** Alternatively we can call addClass() method and pass in the class name which needs to persist. To add multiple classes , we can call addClass() multiple times. Below code snippet uses addClass() method to load user and account classes available in com.hibernate.tutorial package.

Syntax

**Configuration configuration = new Configuration();**

**configuration.addClass("com.hibernate.tutorial.user.class");**

**configuration.addClass("com.hibernate.tutorial.user.account.class ");**

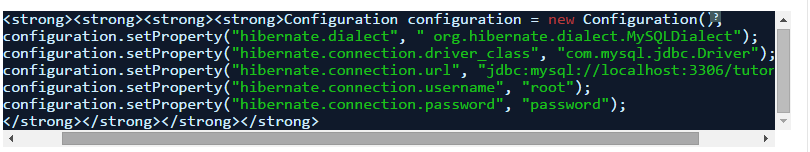
1. **addResource() –** We can call addResource() method and pass the path of mapping file available in a classpath. To load multiple mapping files, simply call addResources() method multiple time.

**Configuration configuration = new Configuration();**

**configuration.addResource("com/hibernate/tutorial/user.hbm.xml");**

**configuration.addResource("com/hibernate/tutorial/account.hbm.xml ");**

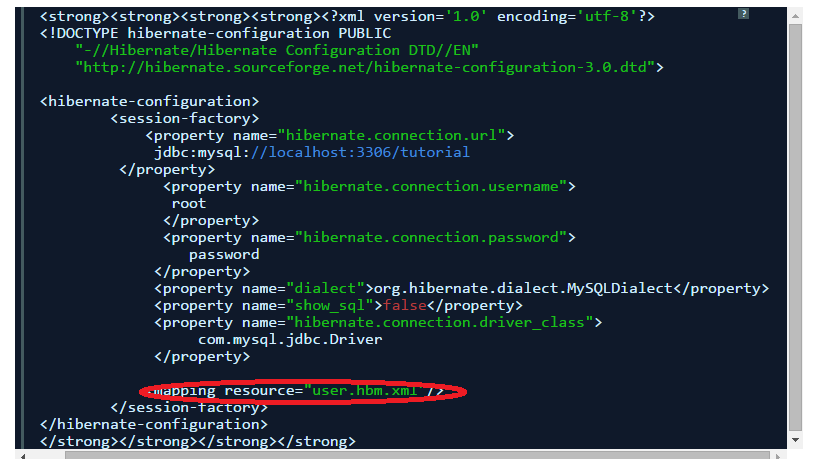
**Note** in the above two methods we provided the hibernate information with respect to the Object now lets see how to provide the database informations



Note: See what are the purpose of the System class

1. **X**ml Configuration example,

**The XML configuration approach is widely used** and Hibernate loads the configurations from a file with name hibernate.cfg.xml from a class path



1. Properties file configuration:
2. Hibernate looks for a file named hibernate. properties file in the class path. Properties file provides the similar functionality as XML file provides with the difference of “ we cannot add a mapping resource in properties file)
3. Syntax:

**hibernate.connection.url = jdbc:mysql://localhost:3306/tutorial**

**hibernate.connection.username = root**

**hibernate.connection.password = password**

**hibernate.dialect = org.hibernate.dialect.MySQLDialect**

**hibernate.show\_sql = true**

1. Few other properties of Hibernate is

* hibernate.format\_sql 🡺Logs the formatted generated SQL on console. Possible values are true or false
* hibernate.hbm2ddl.auto==> this Property has the following values,

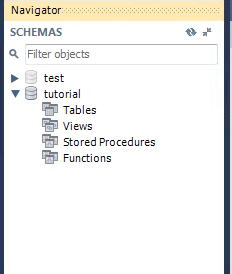
create,

update,

create-drop🡺 drop the tables when session factory is closed and create when application starts based on mapping files.

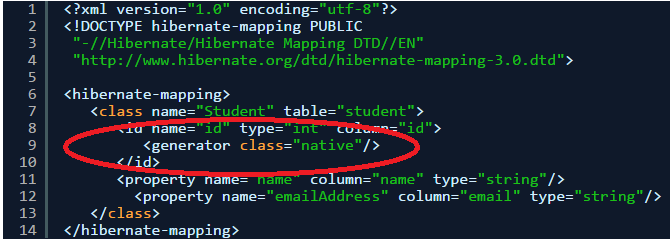
validate 🡺does not create table validates the table against mapping files. Gives errors if mismatch

* hibernate.connection.pool\_size🡺 To limit the number of connections.
* hibernate.connection.driver\_class🡺JDBC Driver class
* hibernate.connection.autocommit🡺To set auto commit mode of connection

1. Schema is the Collection of tables, functions,Views, Stored Procedures. For example look at the below image, in which I created the schema called tutorials.

Note: Database is a collection of the Scemas.

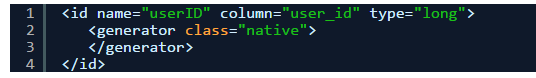
1. Syntax of hbm.xml file



1. Once the session factory object is obtained, configuration object is no longer required.
2. most Commonly used APIs of SessionFactory are
3. close() 🡺this method is used to destroy the session factory after releasing all the resources.
4. getCurrentSession()🡺this method is used to get the current session
5. getStatistics() 🡺 this method is used to get all the statistics of the session factory
6. isClosed() 🡺 this method is used to check is the session factory is closed or not
7. openSession() 🡺 this method is used to get or create a new session
8. openSession(Connection) 🡺this method is used to get or create a session using provided connection object. **This method is not recommended.**
9. Session objects provide a connection with a relational database🡺 that is if we wanted to do any operation(crud) on the database we need session object which physically establishes the connection
10. most Commonly used APIs of Session are
11. beginTransaction() – this method is used to start a new transaction. New transaction object is returned by this method.
12. clear() – this method is used to clear the current session.
13. cancelQuery()- this method is used to cancel the current query execution.
14. close() – this method closes the session and release the resources.
15. connection() – we can get the connection associated with the session using this method.
16. contains(Object) – to verify if the given instance is associated with the session or not.
17. createQuery(String) – this method returns a Query Object for the given HQL.
18. createSQLQuery(String) – this method returns SQLQuery instance of given SQL.
19. delete(Object) – this method deletes the row from the database associated with the given object.
20. flush() – this method flushes the session and updates the state of all associated instances.
21. get(Class, Serializable) - this method returns the persistent object corresponding to the given identifier. If there is no persistent object corresponding to the identifier, it will return null.
22. getNamedQuery(String) – this method is used to get the instance of Query for the given named query defined in the mapping file.
23. getSessionFactory()- this method is used to obtain the session factory which is being used to get the session.
24. getStatistics()- this method is used to get the statistics of the session object.
25. getTransaction()- this method returns the associated transaction.
26. isConnected() – this method is used to check if the session is still connected.
27. isOpen()- this method is used to verify if the session is open or not.
28. isDirty()- this method is used to verify is there are changes pending to be updated in the database or not.
29. Load (Class, Serializable) – this method is similar to get () method, but it throws exception if the persistent object with a given identifier is not found.
30. Merge (Object) - this method will assign a new reference to the given object.
31. refresh(Object)- this method loads the object again from database so in case the database state has been changed directly then object will be in sync of the database.
32. Save (Object) – this method saves the object to the database.
33. saveOrUpdate(Object) – this method either saves or updates the given object.
34. update(Object) – this method is used to reattach the detached object.
35. **Query API** 🡺 is used to execute the Hibernate Query Language (HQL).
36. executeUpdate() – this method is used to execute the delete or update statements on database and returns the number of affected rows.
37. list() – this method returns the result as list
38. Iterator () - returns the results as an iterator.
39. **setBoolean(int , boolean) – used to set the value of a given numbered placeholder.**
40. **setBoolean(Stirng, boolean)- used to set the value of a given placeholder.**
41. setFirstResult( int ) - we can specify the number from which we would want to get the results.

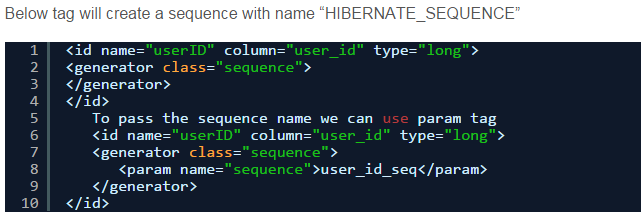
**Primary Key**

1. When we persist Java objects using Hibernate, primary key has to be generated.
2. Primary key uses,
3. To uniquely identify the object
4. To obtain the relationship between the tables.
5. <id> tag is used to specify the identifier in mapping documents, Id element supports below attributes,
6. name – is the name of the property of object which will be used for primary key.
7. type – is the data type of field.
8. column- column name in table to hold primary key.
9. generator class – generator class implementation which will be used by hibernate to generate the primary key.{MS is used to tell which generator class needed to be used to generate the ID}

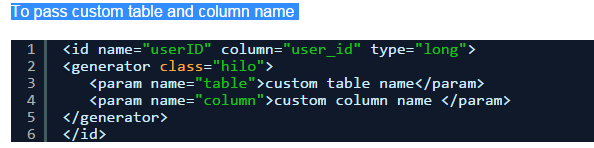


1. different types of generator modules are,
2. **assigned** 🡺 this is the default identifier generation strategy (in case generator tag is not defined). With this approach, its application's responsibility to generate and assign the primary key in identifier property.
3. **identity** 🡺this approach is database dependent which forces us to make sure about the database with which application is interacting supports identity or not before using. In this case database generates the unique value and not by Hibernate.
4. **sequence** 🡺 this approach is database dependent because not all databases supports sequence. This forces us to make sure about the database with which application is interacting supports sequence or not before using. Hibernate gets the next value using the sequence.

If we did not define any sequence in the database, then, hibernate creates a one with name “HIBERNATE\_SEQUENCE”. For auto creation of sequence , hbm2ddl.auto must be enabled. Alternatively, we can pass the name of sequence which has been created in database explicitly.



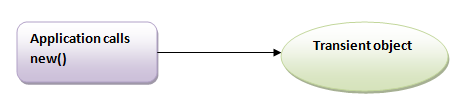
1. increment – this approach is database independent and increments the maximum value by one and use as identifier .In case there is no row in database , then for the very first row ,value will be
2. hilo (high/low ) – this approach is database independent and a very efficient way of generating identifier. This approach requires a separate table and a column with default table name as “hibernate\_unique\_key” and column name as “next\_hi” . We can pass custom table and column name as well.



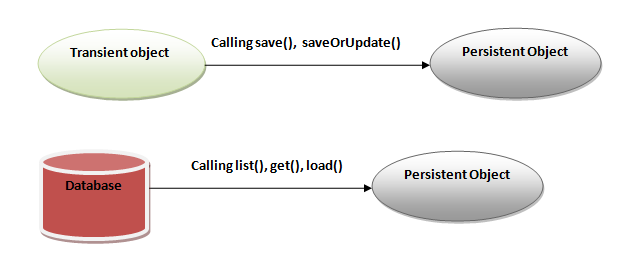
# Hibernate Advance Mapping

# Working with Objects in Hibernate

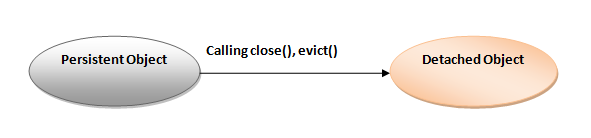
1. Object States and Life Cycle
2. Transient – any object which has just instantiated using new keyword and has not associated with any session.



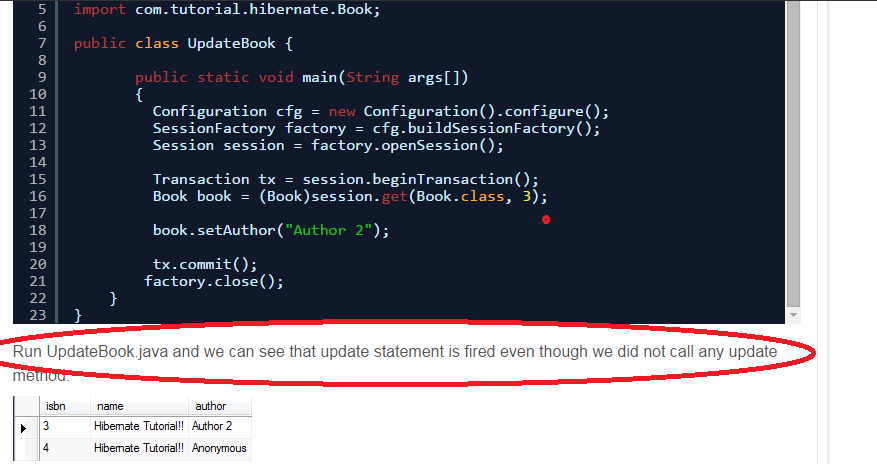
1. Persistent - Persistent instances are the objects that are currently mapped with a session and has a corresponding row in a relational database. Any object can be in persistent state if object has been loaded by database or has been saved.



1. Detached- Detached instances are the objects that are mapped to the session or persistent context but session is closed later. In other word , objects whose mapped session is closed.



1. We can call **save() method** to save an object in to the database and also this operation makes a transient object to persistent object
2. If there are other objects mapped from the primary object, they are saved when we commit the transaction or flush the session.
3. Hibernate does provide two ways to get the persistent object
4. **Load ()** 🡺does not make a call to the database instead gives a proxy object with an identifier value populated only and other properties remain uninitialized. In case row corresponding to given identifier does not exists, **ObjectNotFoundException** will be thrown.
5. get()- method calls the database to find the object associated with the given identifier. In case object does not found, returns null.
6. **Most of the developers assumes that “update()“ method of session object is used to update the state of an on object but this is wrong.** update method is designed for something else ( we will cover this later in this chapter). It is important to understand and know that states of an object automatically gets updated in database when,
7. On calling session.flush()
8. On calling transaction.commit()



1. To delete any persistent object, we can call delete() method passing the object to be deleted. **Delete() method also changes the state of a persistent object to transient state.**
2. **Reattach a Detached Object OR modifying detached Object🡺here we will see the actual use of the update method**

In any of web application we may need to reattach the detached object. Think of the situation when back end system loads the object from database and closes the session. The object gets modified by another layer and requires update in the database. To reattach the detached object there are two options –

1. **Update() 🡺** as mentioned earlier, this method is not designed to update the state of an object instead it is designed to reattach the detached objects**. If session already contains a persistent object with same identifier, it will throw an exception.This method returns void**
2. **Merge() 🡺** merges the modifications with the persistent object with same identifier and returns the reference of new object. We should use merge if we want to update the changes without considering the state of a session. The returned object is part of persistent context and tracked for any changes where as the passed object will not be associated with the session.
3. **saveOrUpdate🡺** Hibernate does provide a method saveOrUpdate() which save the object if there is no row available for given identifier and update the row if found. In case object with the same identifier is available in session, it will throw an exception.

# Hibernate Query Language (HQL)

1. Hibernate Query Language is a similar to SQL but in terms of Object🡺 that query will not have the columns and table details rather it contains class and member variables details.
2. We may want queries with where clauses, conditions , aggregate functions etc in our application and HQL does support all these features
3. **HQL are case insensitive. Using HQL keeps our application code database independent.**
4. We can create a query object, using session object.

Query query = session.createQuery(“ from Object ”);

From is the only mandatory part of HQL.. It tells hibernate from where to pull the data.

**Aliases**

1. We can assign an aliases to the objects in HQL by just adding an alias next to object OR using “as” keyword

Syntax:

Query query = session.createQuery(“ from Book **as bookAlias** ”);

**or**

Query query = session.createQuery(“ from **Book bookAlias** ”);

**Select Clause**

1. There is no need of select clause but if we want to get selected properties only we need to use select clause providing the list of comma separated name of the object properties .

**Query query = session.createQuery(“ select alias.name from Book as alias ”);**

1. To get a list of an object using select statement , we can use constructor . For example,

**Query query = session.createQuery(“ select new Book(name, price) from Book ”);**

**List<Book> books = query.list();**

**where Clause**

1. Query query = session.createQuery(“select name from Book where isbn=’123’ ”);

**Supported Operators**

1. Logical Operators - AND , OR , NOT
2. Equality Operators - =, !=
3. Comparison Operators - < , > ,<= ,>=, LIKE, NOT LIKE , BETWEEN , NOT BETWEEN , IS NULL , IS NOT NULL , IN

**Pagination**

1. HQL supports pagination where we can specify the starting record number and number of records to fetch. To do so we can use setFirstResult and setMaxResults methods.

For example: Below query will fetch 15 records starting from 5th record.

**Query query = session.createQuery(“select name from Book ”);**

**query.setFirstResult(5);**

**query.setMaxResults(15);**

1. Instead of hard coding the variables values, we can bind the values at run time. We can bind the parameters using,
2. Name 🡺also known as named parameter binding
3. Position 🡺 also known as positional parameter binding.

Let’s see them in detail below,

### ****Named Parameter Binding****

1. Using named parameter binding , query becomes more readable and same parameter can be reused multiple times. To use named parameter we need to use “**:name”**

Syntax For :

**Query query = session.createQuery(“select name from Book where name = : bookName ”);**

**query.setString(“bookName”, “XYZ”);**

1. **Second way,** Similar to traditional JDBC Prepared Statement style , we can use ? as a place holder ( 0 based)

Syntax: **Query query = session.createQuery(“select name from Book where name = ? ”);**

**query.setString(0, “XYZ”);**

**Functions**

1. Functions like avg() , sum(), count() , max() , min() , distinct are supported.

**Syntax: String hql = “ select sum(book.price) from Book as book”;**

**Query query = session.createQuery(hql);**

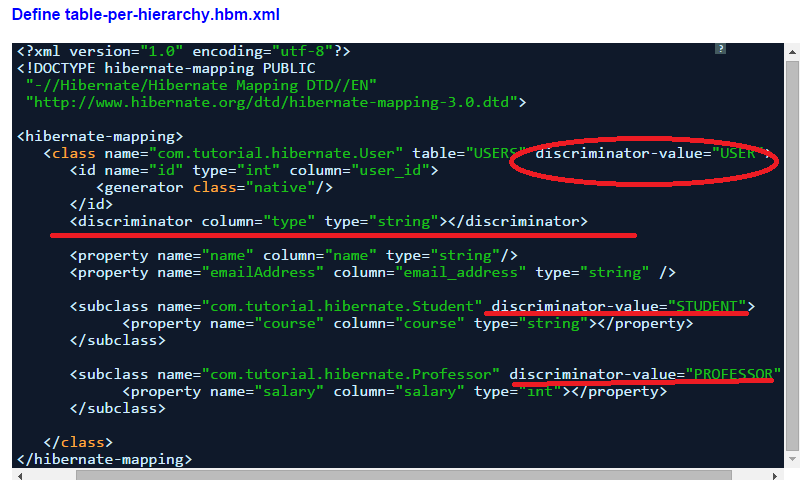
1. We have **“order by”** for sorting.

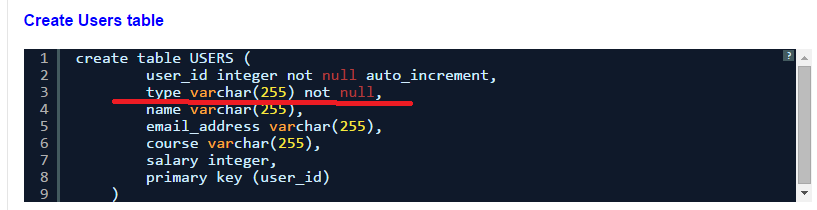
# Inheritance Mapping in Hebernate

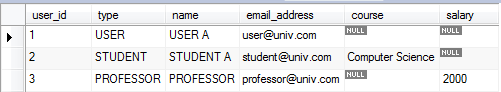
1. **Java Object supports Inheritance where as Relational database does not.**
2. Hibernate supports four approaches to address this gap.
3. Table per Class Hierarchy
4. Table per Subclass
5. Table per Concrete Class with Implicit Polymorphism
6. Table per Concrete Class with Union

**Table Per Class Hierarchy**

1. As the name suggests, there will be a single table for the entire hierarchy. With this approach, all columns corresponding to the properties in subclasses and main class are included in the table
2. **To determine the type of subclass, discriminator column is created in the table**. This column will not have any property in object hierarchy.



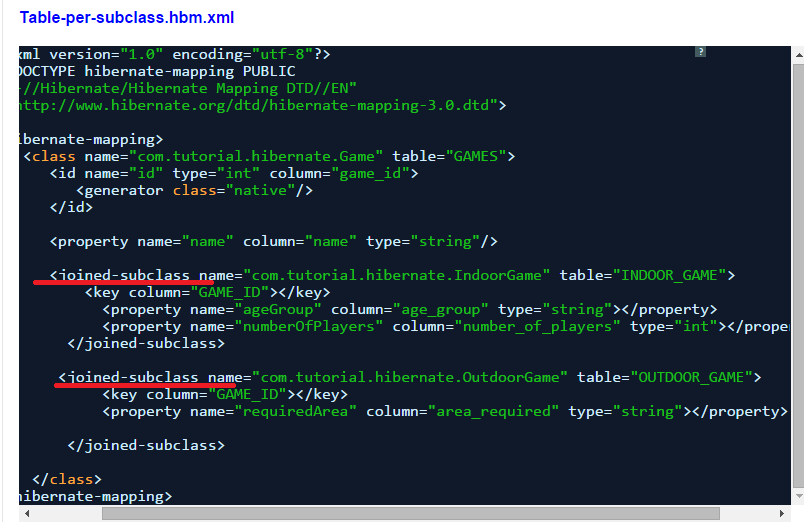


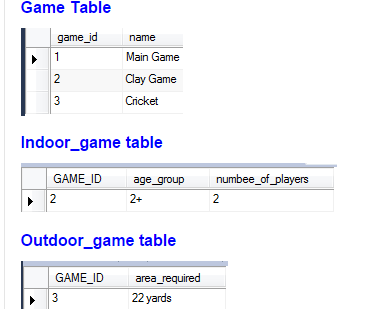
1. 

**Table per Sub Class**

1. In this approach for each and every class (abstract, interface etc ), there will be a separate table and a foreign key- primary key relationship between base and subclasses
2. **Instead of <subclass> , we will use <joined-subclass> tag.**

Example:





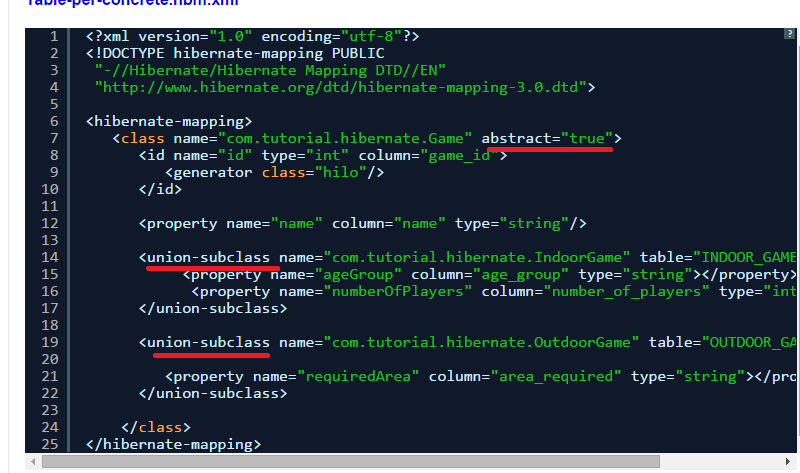
**Table Per Concrete Class with Implicit Polymorphism**

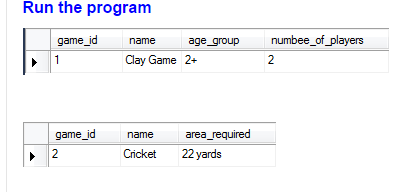
1. **In this approach there will be one table for all concrete (non abstract ) subclass and will contains the column of base class**. Yes you are right – **this approach has a redundant columns.**

**Table Per Concrete Class with Union**

1. This approach works even when base class is abstract or interface (no separate table is needed for base class). If the base class is concrete , there will be table corresponding to the base table as well. We need to specify abstract=true if base class is abstract or interface.
2. **<union-subclass>** tag is used to specify the sub classes.

**Syntax:**





# Component Mapping in Hibernate

1. **Java supports composition in the form of has-a relationship**. Composition is stronger association where the contained object has no existence of its own. For example Person has an address, Person has a job . In these examples job and address have no existence without Person.
2. **We may want to store contained objects like Job, Address etc in the same entity table (person) as a value type and Hibernate does support this feature using <component> tag.**

**Note:**

**Most important thing is – contained object is not stored in a separate table instead it will be stored as a value type in table corresponding to contained object**

**Attributes of <component> element.**

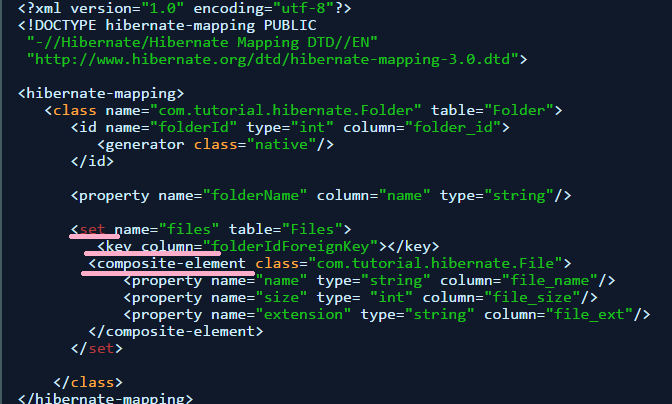
1. Below are the details of most commonly used attributes.
2. **access** 🡺 is used to specify the property access strategy. Properties can be accessed using “field” (no need of get/set) or a property (using get/set) . Possible values are filed and property.

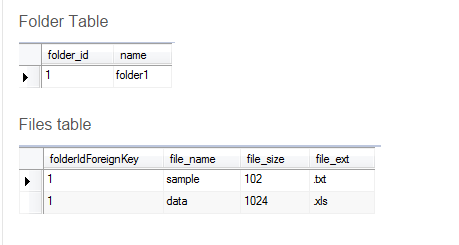
**It is very important to understand that the components have no existence of their own and that is the reason they do not have any primary key.**

# Collection Mapping in Hibernate

1. **If the component is single instance, we can add it as a value type in the same table, but what if the component is a collection? In this case we would need a separate table known as collection table. Again, as these are component, there will be no primary key** (identifier associated with the rows).
2. Types of the collection supported by hibernate,
3. **java.util.Set**🡺 does not allows duplicate and order is not maintained. Is mapped to <set> element of hibernate. Implementation should be HashSet
4. **java.util.SortedSet**🡺does in memory sorting. is mapped to <set> element of hibernate and use sort attribute to configure comparator implementation should be TreeSet
5. **java.util.List** 🡺 allows duplicates and maintains insertion order. is mapped to <list> element of hibernate and use list-index attribute to preserve the index (position). implementation should be ArrayList
6. **java.util.Collection**🡺 allows duplicates and does not maintains insertion order. Is mapped to <bag> or <idbag> element of hibernate. implementation should be ArrayList .
7. **java.util.Map**🡺 is a key value pair. is mapped to <map> implementation should be HashMap.
8. **java.util.SortedMap**🡺 is a key value pair and performs in memory sorting. is mapped to <map> and use sort attribute to configure comparator. implementation should be TreeMap.
9. **Component classes will not have its own identifier and should implement equals and hashcode() method in order to make the uniqueness of the objects** as some of the java collections like set does not allow duplicates.
10. to configure the collections of components (objects ),Hibernate provides 2 ways,
11. “<composite-element>” which can be used in a collections tag like <set>, <map> etc.
12. Other approach is to use association tags like one to many, many to one etc
13. Let’s see the collection implementation by using the <composite-element> tag

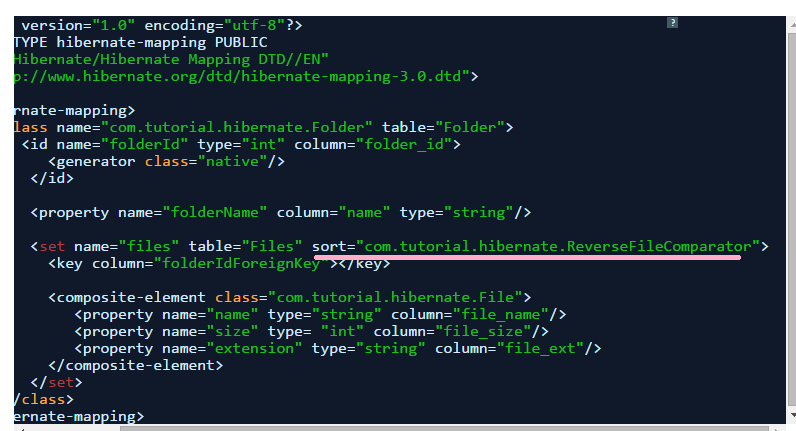






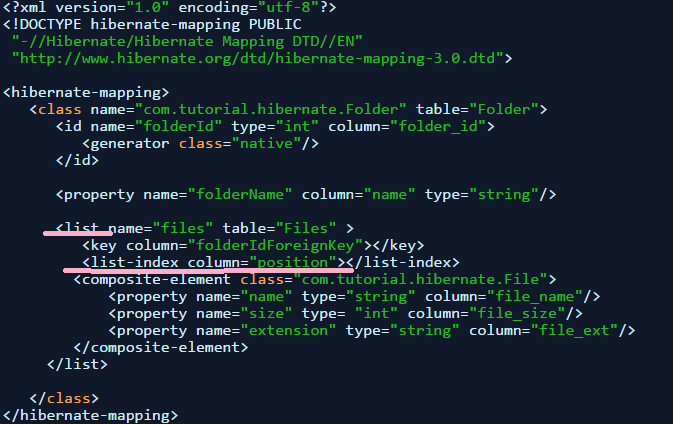
### ****13.3.2. SortedSet Implementation****

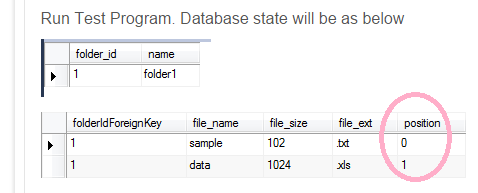
1. We can use SortedSet interface with TreeSet implementation if we want to have the set sorted. **File class has to implement Comparable interface** in order to get it added in TreeSet.
2. In a <set> tag we can use sort attribute to,
3. specify the Comparator class which will be used to sort the set when the files will be retrieved.



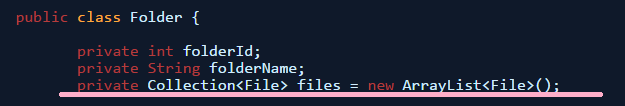
### ****13.3.3 List Implementation****

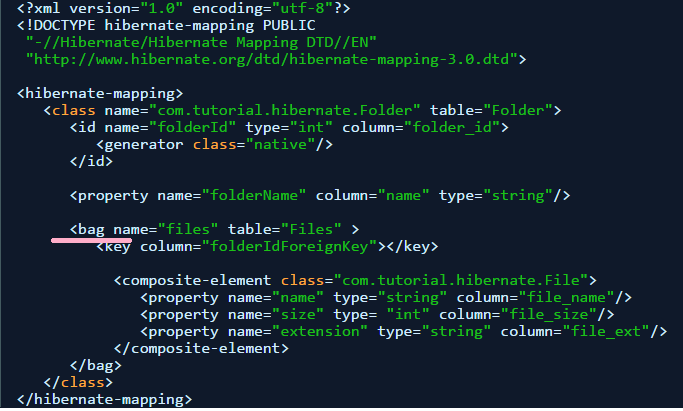
1. Since list is ordered collection, we need to specify a column name in <list-index> attribute to store the index of the list elements.

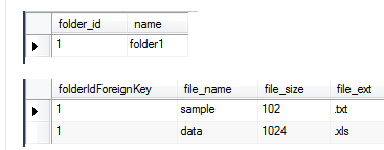




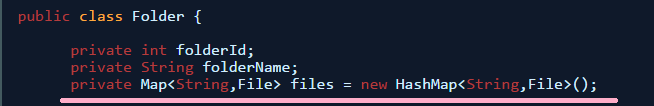
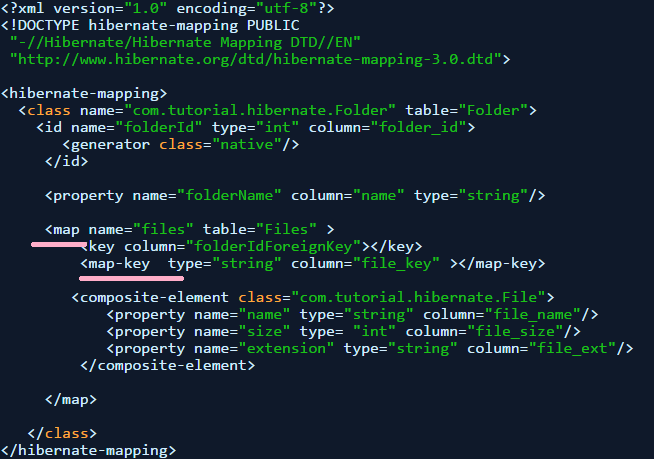
### ****13.3.4 Collection Implementation****

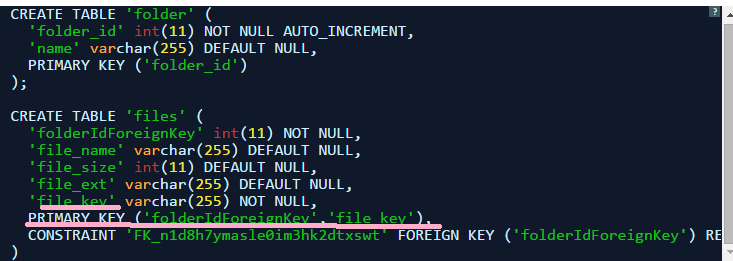
1. To support unordered and duplicated values , we can use Collection interface and ArrayList as an implementation . Hibernate does provide <bag> and <idbag> tags to map the Collections.
2. 





### ****13.3.5 Map Implementation****

1. Map is a key value pair and hibernate does provide <map> tag to support java.util.Map. HashMap should be used as an implementation Folder.java
2. 
3. 

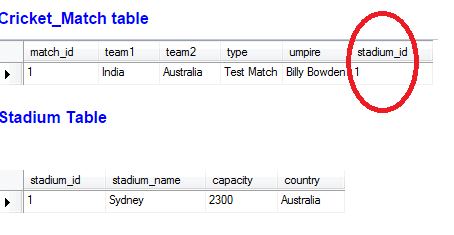


### ****13.3.6 SortedMap Implementation****

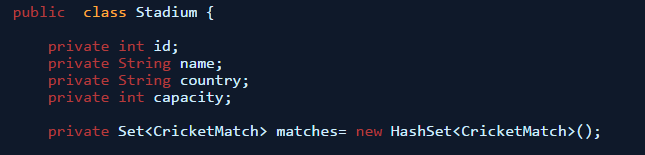
# Association Mapping in Hibernate

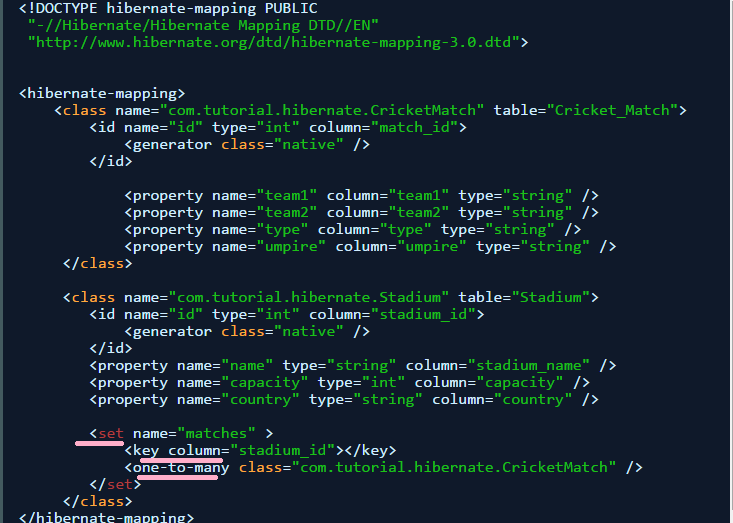
1. **Here is the place where we will see One to many mapping such things and all**
2. As we know that value type objects ({which uses the <component-element> tag}) life span is dependent on the parent object
3. We may have a collection of objects which are not value type instead are shared objects and in such case their life span cannot be dependent on the parent object. In such cases, the contained objects will have their own identifier and hence their own lifecycle.
4. Types of association mapping.
5. Many to One
6. One to Many
7. Many to Many
8. One to One

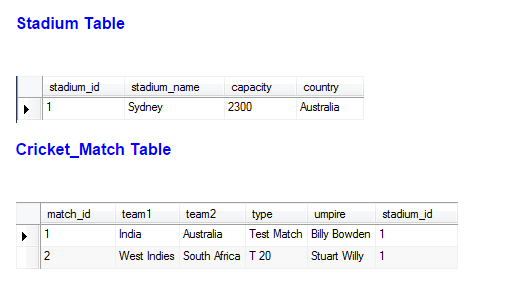
**14.2 Many to One**

1. association-mapping.xml - <many-to-one> tag is used
2. 
3. 

**One to Many Association**

1. 





**One to One mapping**

1. We discussed **many to one mapping and adding unique=true can be equivalent to one to one mapping.** Hibernate does provide one to one tag also, which works on sharing primary keys concept.

for example🡺**One Person can have one and only one Passport**

<class name="com.tutorial.hibernate.Person" table="Person">

<id name="id" type="int" column="person\_id">

<generator class="native" />

</id>

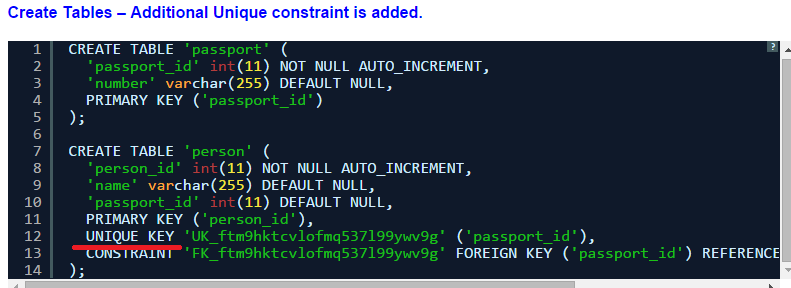
<property name="name" column="name" type="string" />

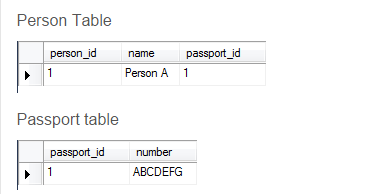
<many-to-one name="passport" class="com.tutorial.hibernate.Passport" column="passport\_id" **unique="true">**

</many-to-one>

</class>

1. **in the above example pink color indicates the properties of the class**



1. 

**Many to Many mapping**

1. **Many to Many mapping requires an additional table (also known as mapping table) which contains the keys of both the tables**

# Hibernate Cascade and Fetching Strategy

**15.2 Cascading Object State**

1. The whole concept of parent child relationship is that one object should take care of persistence of another object internally and application programmer need not write additional code to manage the persistence

# Hibernate Interceptors and Events

1. Hibernate is all about Entity persistence and quite often we would want to intercept the request or perform some tasks when state of an object changes.
2. With the help of interceptors , we get an opportunity to inspect the state of an object and if needed we can change the state as well.To support this Hibernate does support the concept of interceptor and provides a Interface and a implementation of Interface. Developers can either directly implement the Interface or may extend the implementation.

**16.2 Interceptors**

### ****(1)**Interceptor **Interface****

1. We can directly implement org.hibernate.Interceptor available in Hibernate. There are several methods available but most commonly used are,
2. **onSave()🡺** this method is called before on object is getting saved and we can modify the state of an instance. Modified state will be persisted.
3. **onDelete()🡺**is called before on object is getting deleted.
4. **onLoad()🡺** is called before object initializes and when this method is called passed entity will be uninitialized.
5. **findDirty()🡺** is called when flush method of session is called.
6. **preFlush()🡺** is called before flush method is called (before database is updated).
7. **postFlush() 🡺** is called after flush method is completed (after database is updated)

### ****2  EmptyInterceptor****

1. Hibernate comes with an empty implementation of Interceptor interface under package org.hibernate with name EmptyInterceptor. It is always advisable to extend EmptyInterceptor instead of implementing Interceptor interface as we would just need to implement the required method and not the entire interface.

### ****3  Configure Interceptor****

1. Interceptor can be configured at a session level and at a session factory level.
2. **At a Session Level**🡺 Pass an instance of Interceptor while opening a session. In hibernate 3 you can use SessionFactory.openSession( Interceptor) whereas in hibernate 4 SessionFactory.withOptions() .interceptor(Interceptor()).openSession();

This way we can get a control at a session level and can enable interceptor for a desired sessions only.

1. **At a Session Factory Level** 🡺Set an interceptor in a Configuration Object. Doing this , interceptor will be enabled for all the session for the session factory. Like,

**Configuration.setInterceptor(Interceptor)**

**Note: Events part and the Programs based on Interceptors and Events we will see later.**

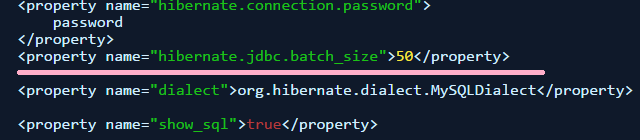
# Native SQL and Named Queries in Hibernate

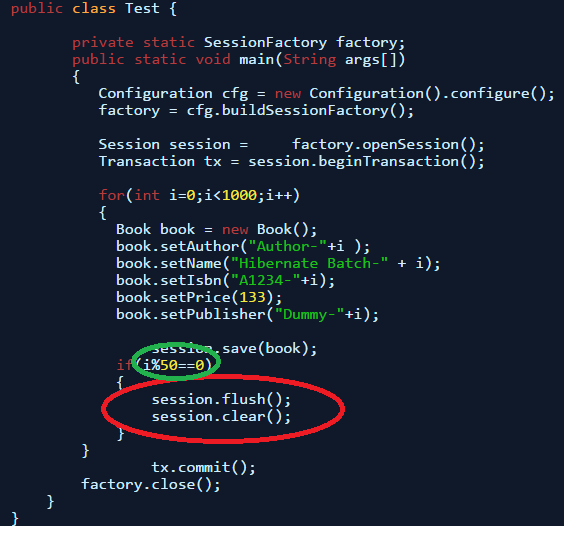
1. There might be some scenarios where we would want to use database specific functions or data types etc. To support such scenarios, Hibernate provides the support for native SQL as well.
2. For HQL we use **createQuery(SQL)** method for HQL where as for native SQL we have to use **createSQLQuery(SQL)** method of session API to get the **SQLQuery** instance.
3. **Native SQL uses table names not class names in query.**

# Hibernate Batch Processing

1. programs where we need to update the bulk data in the database or may need to insert huge number of records. In such scenarios we need to flush the session at a regular interval to make sure that we did not hit OutOfMemoryException because hibernate first level cache is session cache.
2. **Flushing and clearing the session at a regular intervals will persist the data and will reuse the memory and hence we can avoid the out of memory exception. This approach is known as batch processing.**
3. For Batch processing in hibernate, we need to set the JDBC batch size in hibernate cfg file using hibernate.jdbc.batch\_size property. The ideal value of this property is between 10 to 50
4. **Make sure you flush and clear the session after the insert and update of records equal to the number you configure in hibernate.jdbc.batch\_size property.**

**Below is the program for inserting the 1000 records into database see below image, after inserting 50 records flush and clear methods are called.**

1. 



# Hibernate Criteria Queries

# Native SQL and Named Queries in Hibernate

**Data Type Mapping**

