Hibernate Basics

INTRODUCTION

ORM

(Object Relational Mapping with Hibernate)

OBJECTIVES

- *Object Persistence and Object Relational Mapping*

LEARNING OBJECTIVES

At the end of this lesson, you will be able to:

- Define Object Persistence
- Describe Limitations of JDBC
- Discuss ORM benefits
- Discuss Object Relational Mismatch
- Hibernate as ORM solution provider
- Hibernate Advantages

Object Persistence

- The state of an object can be permanently stored even after the execution of the program.
- Persistence is the mechanism by which the data outlives the process that created it.
- we would like the state of (some of) our objects to live beyond the scope of the JVM so that the same state is available later.
- There are many ways to persist data in java.
  - JDBC
  - Serialization
  - File I/O
  - Object Database
  - XML Database

Limitations of JDBC

- A lot of programming overhead is involved when JDBC is used for persisting data into database , Hence not suitable for large projects.
- Programmer must hardcode the transaction and concurrency code in the application.
- A lot of boilerplate code related to handling JDBC connection , closing connection and handling exceptions is repeatedly written by the developer in all modules.
- If the database changes in future then the sql code written in Data access Layer must be replaced with different implementation of data access code.

ORM (Object Relational Mapping)

- It is a technique that lets you query and manipulate data from a database using an object oriented paradigm.

- It is a way of integrating Object Oriented Programming Language capabilities with the relational databases like Oracle or SQL Server.

ORM Advantages

- It fits your natural way of Coding in Object oriented style of programming for data persistence.
- It abstracts the Database Systems , So you can change database whenever you want without changing the SQL queries.
- Database operations and manipulations becomes easier as many aspects of the application like transaction management can be handled by the ORM library/implementation.

Object Relational Impedance Mismatch

- Also called as Paradigm Mismatch
- It means Object models and Relational models do not work very well together

Object Model	Relational Model
Data is represented as Inter Connected Graph of Objects	Data is represented in Tables , in the form rows and columns
More Granular	Less Granular
OOP Language like Java defines Object identity and Object Equality , two different notion of sameness.	RDBMS defines exactly one notion of sameness i.e. the Primary Key.
Inheritance is a commonly used feature in Object models	No Standardized support for inheritance among tables across RDBMS systems
Data navigation in java is done by navigating from one association to another walking the object network.	Data navigation is done through JOINs and Queries.

Introduction to Hibernate

- A Framework which simplifies the development of java application by providing effective mechanism for interacting with databases and persisting data.
- Hibernate is the ORM tool given to transfer the data between a java (object) application and a database (Relational) in the form of the objects.
- An ORM tool provides a programming technique that maps the object to the data stored in the database and offers better and easier mechanism for manipulation and access of data.
- Hibernate Internally uses JDBC API to interact with the database.

Hibernate Advantages

- Open Source and Light weight
- Fast Performance
- Database Independent Queries
- Automatic Table Creation
- Simplifies complex joins
- Provides Query Statistics and Database Status

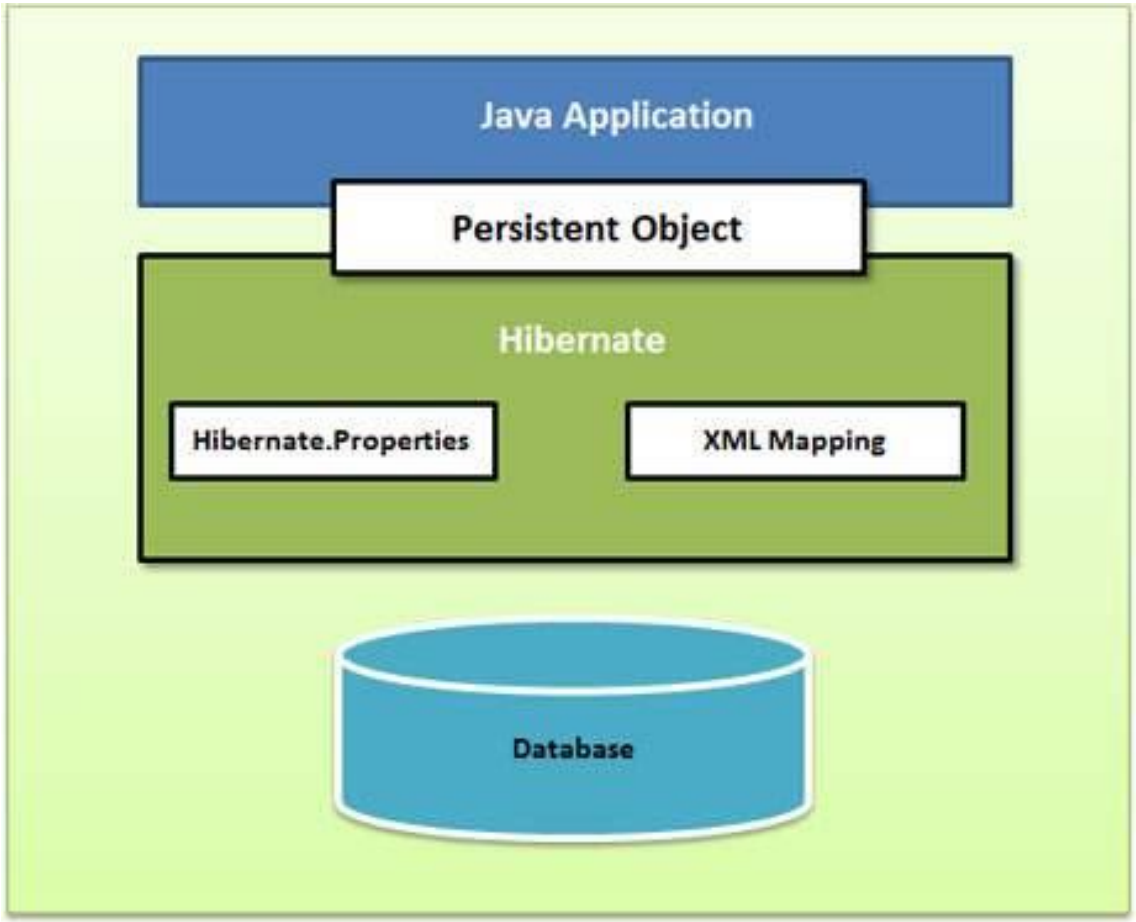
Hibernate Architecture

- Hibernate makes use of the database and configuration data to provide persistence services to the application.
- The Configuration object in Hibernate represents the configuration properties which provides two key components

Database Connection: This is handled through one or more configuration files supported by Hibernate. These files are **hibernate.properties** and **hibernate.cfg.xml**.

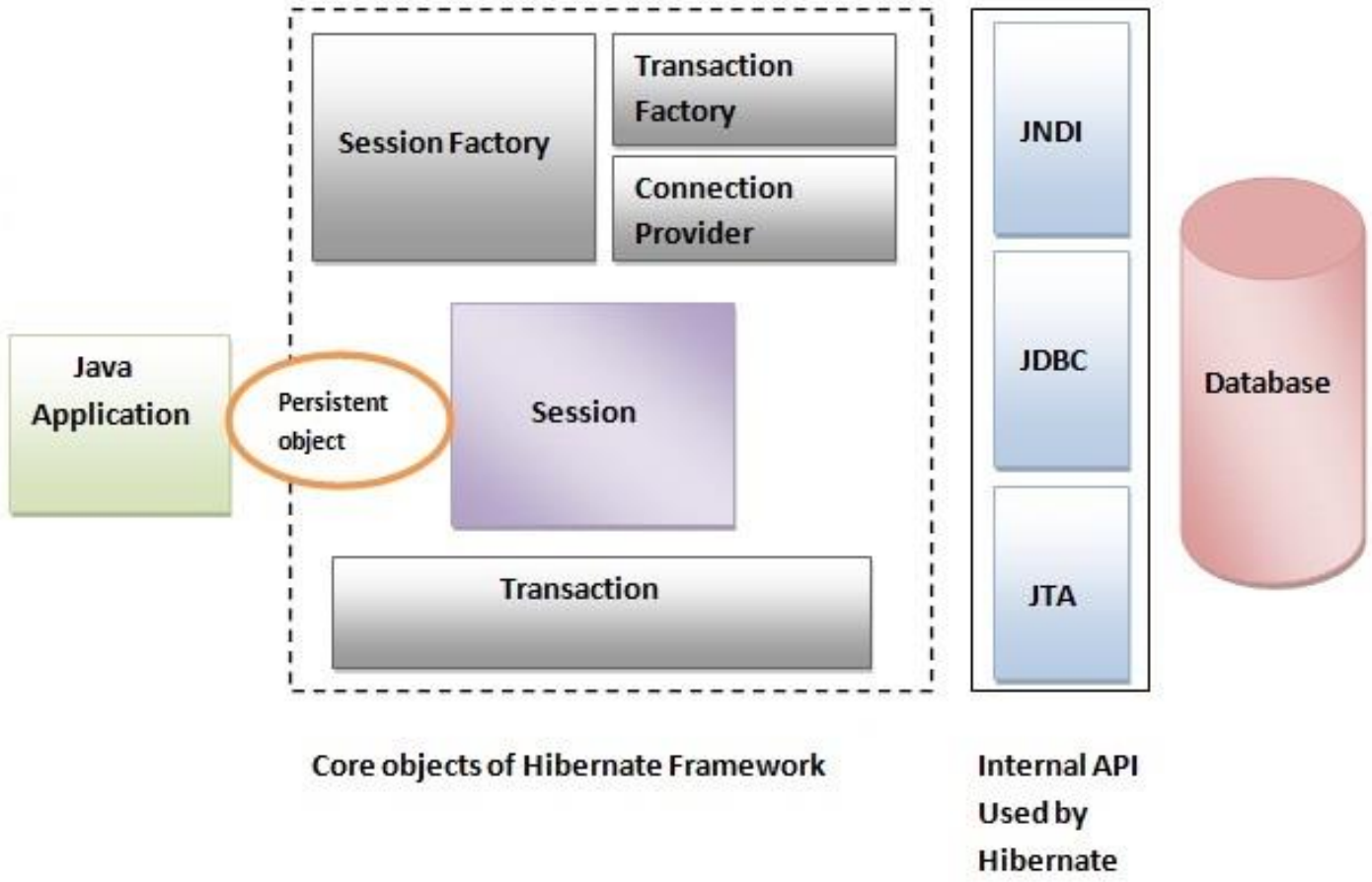
Class Mapping

This component creates the connection between the Java classes and database tables.



Hibernate Detailed Architecture

- **SessionFactory :** A factory of Session
- Session : Provides and interface between the application and data stored in database.
- **Connection Provider** : A factory of JDBC connections abstracting the application from DriverManager or datasource
- **Session** : A Session is used to get a physical connection with a database
- **Transaction** : Transactions in Hibernate are handled by an underlying transaction manager (JDBC or JTA)



Steps-Developing and running Hibernate Application

1. Create a java project
2. Add Hibernate Jar files
3. Create the Persistent classes
4. Create the mapping file for Persistent class
5. Create the Configuration file
6. Create the class that retrieves or stores the persistent object
7. Run the application

Hibernate POJO Class(Entity Class)

- POJO is a simple java file which does not extend or implement any other class or interface.
- POJO classes represent the entity objects which are required to be persisted by hibernate .
- The POJO class contain private properties variables, and for each property a setter and a getter

```
public class  MageTrainee
{
private int traineeID;
private String name;
private String Stream;

. .   .   .

//Public Getters and Setters



 }
```

Hibernate Mapping File (XML)

- The mapping file contains the mapping metadata for the java entities and their corresponding relational Tables.

```
<hibernate-mapping>
   <class name="MageTrainee" table="MAGE_TRAINEE">
         <id name="traineeId" column="TRAINEE_ID">
              <generator class="assigned"/>
         </id>
         <property name="name" column=”NAME" />
         <property name="stream"/>
</class>
</hibernate-mapping>
```

Hibernate Configuration and Session

```
//creating configuration object
Configuration cfg=new Configuration();
cfg.configure("hibernate.cfg.xml");

//creating seession factory and session object
SessionFactory factory=cfg.buildSessionFactory();
Session session=factory.openSession();
```

Hibernate Transaction

```
//creating transaction object
    Transaction t=session.beginTransaction();
    MageTrainee trainee=new MageTrainee();
    trainee.setTraineeID(101);
    trainee.setName("Mohan");
    trainee.setStream("Java");

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

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

First Hibernate Application Demo

Demo : FirstHibernateDemo

Hibernate ID Generator Classes

The <generator> sub element of id used to generate the unique identifier for the objects of persistent class.

Following are some of the common generator classes supported by Hibernate

- Assigned
- Increment
- Sequence
- Native
- Identity

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

    .....

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

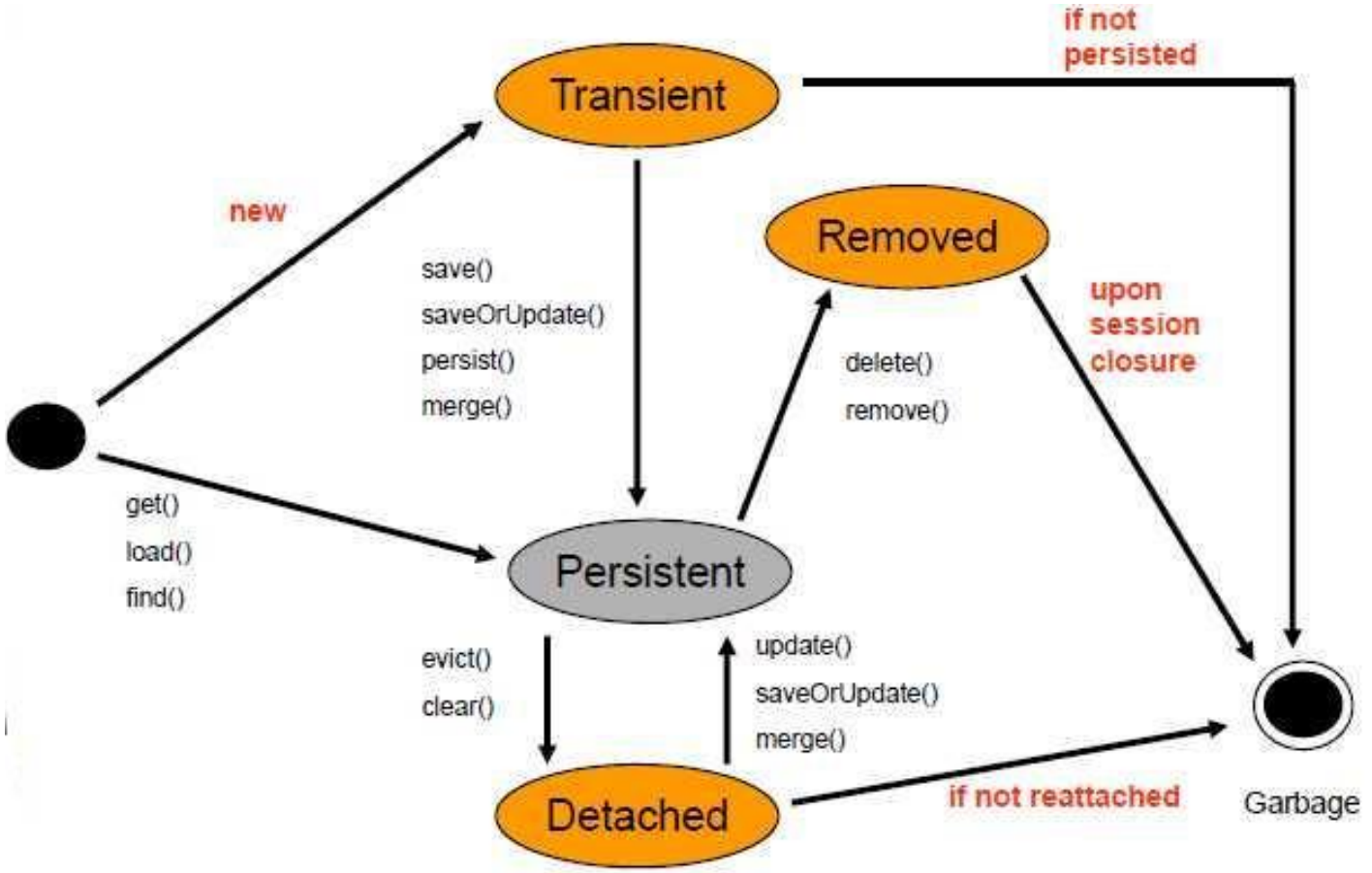
SQL Dialects in Hibernate

- To connect any hibernate application with the database, you must specify the SQL dialects.
- Many Dialects classes are defined for RDBMS in the org.hibernate.dialect package.
- Few of them are given below.

RDBMS	Dialect
Oracle (any version)	org.hibernate.dialect.OracleDialect
Oracle9i	org.hibernate.dialect.Oracle9iDialect
Oracle10g	org.hibernate.dialect.Oracle10gDialect
DB2	org.hibernate.dialect.DB2Dialect
MySQL	org.hibernate.dialect.MySQLDialect

Life cycle states of Hibernate Entity

The lifecycle methods and interfaces of JPA packages are implemented by Hibernate Entity Manager.



Transient Object State

- Any object which is created using the new() operator for the first time is said to be in transient state.
- When the object is in transient state , it will not hold any value.
- We will be using any one of the following (Save(), saveOrUpdate()) methods to persist the transient object.

t

```
Employee e1=new Employee();
e1.setEmpno(117);
e1.setName("Shankar");
```

Transient Object

Persistent Object State

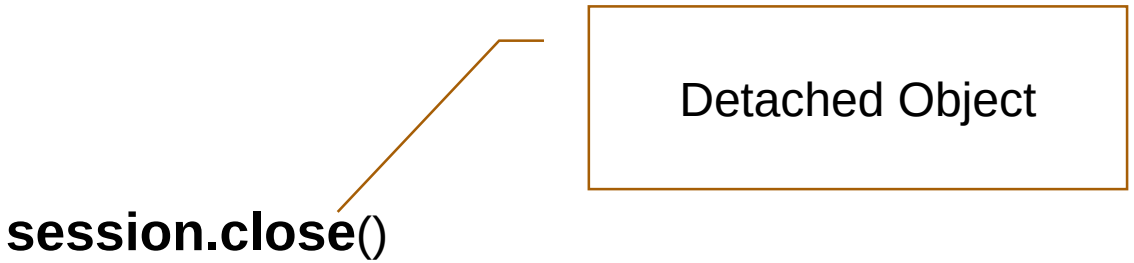
- Any (POJO) object associated with hibernate session is called as Persistent Object.
- represents a row / record with a identifier value.
- Any object in transient state can be made into persistent state by associating with a Session

```
Employee e1=new Employee();
e1.setEmpno(117);
e1.setName("Shankar");

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

t.commit();//transaction is commited
```

Persistent Object

Detached Object State

- When the (POJO) object is removed from the hibernate session, we can say that the object is in detached state.
- Any modifications made to the object are not saved to the database.
- Any changes made to the detached objects are not saved to the database
- The detached object can be reattached to the new session for saving.



Hibernate Annotations

- We can map the classes without using the XML file by Hibernate Annotations
- provides the metadata for the object as well as Relational Table mapping.
- helps the developer to understand the structure of the table and related POJO class while developing the code
- this feature is supported from jdk5.0 and above.
- Hibernate Application can also be developed using Annotations

Hibernate Annotations

- Annotations which can be used to create hibernate application.
- @Entity , @Id , @Table
- The package javax.pesistence.* contains all JPA annotations.
- Main advantage of using hibernate annotation
  - does not require hibernate mapping files (.hbm) to be created
  - it gives the meta data

Creation of Hibernate Application using Annotation

- There are four step approach for creating an Hibernate Application using Annotation.

- Step 1: Create a Java Project and add the required jar files for database(Oracle or MySQL).
- Step 2: Create a Persistent class for it.
- Step 3: Add the mapping of Persistent class in configuration files.
- Step 4: Create the class which stores or retrieves the persistent object

Step 1 Adding of jar files for Database and annotation

- The following jar files must be added in the build path of the project.
- Jar file required for Oracle
  - ojdbc6.jar

- Jar files required for annotations
  - hibernate-commons-annotations.jar
  - ejb3-persistence.jar
  - hibernate-annotations.jar

Step 2: Creation of Persistent class

- @Entity is the annotation that identifies the class as an entity .
- @Table defines the Table name where we can store the Entity details. If not defined class name will be taken as Table name.
- @Id used as an identifier for entity.
- @Column defines the attributes / field name where we can store the Entity details. If not property name will be taken as column name.

Step 3: Add the mapping of Persistent class in configuration files

```
1  <?xml version='1.0' encoding='UTF-8'?>
2  <!DOCTYPE hibernate-mapping PUBLIC
3   "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
4   "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
5
6   <hibernate-mapping>
7       <class name="com.manipal.hb2.Student" table="Student_det">
8      <id name="rollno">
9       <generator class="assigned"></generator>
10     </id>
11     <property name="name"></property>
12    </class>
13  </hibernate-mapping>
```

Step 4: Create the class which stores or retrieves the persistent object

```
public class ManageEmployee {
   private static SessionFactory factory;
      ----

public static void main(String[] args)  {
    ManageEmployee me = new ManageEmployee();
    /* Add few employee records in database */
    Integer empID1 = me.addEmployee("Zara", "Ali", 1000);
    Integer empID2 = me.addEmployee("Daisy", "Das", 5000);
    Integer empID3 = me.addEmployee("John", "Paul", 10000);
```

Access Type

- The default access type of class hierarchy is defined by the position of @Id
- If the annotations are present on the field, that field alone will be considered for persistence.
- In some cases we need to :
  - force the access type of the entity hierarchy
  - override the access type of a specific entity in the class hierarchy
  - override the access type of an embeddable type
- We use @Access annotation to force the access type of a class

```
@Access(AcessType.PROPERTY)
public class Student {
   private String name;
   public String getName() { return name; }
   public void setName() { this.name = name; }

   private hashCode; //not persistent
}
```

Property Annotations

- @Column annotation is used for mapping the property defined in the class.
- Some of the property level annotations for properties are :
  - @Basic
  - @Version
  - @Lob
  - @Temporal
- If a property is not annotated, the following rules apply:
  - If the property is of a single type, it is mapped as @Basic
  - if the type of the property is annotated as @Embeddable, it is mapped as @Embedded
  - if the type of the property is Serializable, it is mapped as @Basic in a column holding the object in its serialized version
  - if the type of the property is java.sql.Clob or java.sql.Blob, it is mapped as @Lob with the appropriate LobType

Hibernate Types

- JPA defines five types of identifier generation strategies:
  - AUTO - either identity column, sequence or table depending on the underlying DB
  - TABLE - table holding the id
  - IDENTITY - identity column
  - SEQUENCE - sequence
  - identity copy - the identity is copied from another entity

Mapping Embeddable Types

- We can declare an Embedded component inside an entity and override its column mapping.
- @Embeddable annotation is annotation used for annotating class level annotation.
- @Embedded and @A ed property to override t

```
@Embeddable
public class Address implements Serializable {
    String city;
    Country nationality; //no overriding here
}
@Embeddable
public class Country implements Serializable {
    private String iso2;
    @Column(name="countryName") private String name;

    public String getIso2() { return iso2; }
    public void setIso2(String iso2) { this.iso2 = iso2; }


    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
    ...
}
```

Composite Keys

- Composite Primary Keys can be defined using various methods

  - use a component type to represent the identifier and map it as a property in the entity. Ensure that component type must be serialized.
  - map multiple properties as @Id properties
  - map multiple properties as @Id properties and declare an external class to be the identifier type.
  - each property name must be the same, its type must be the same as well if the entity property is of a basic type

Generated Keys

- The JPA specification supports 4 different primary key generation strategies.
  - GenerationType.AUTO
  - GenerationType.IDENTITY
  - GenerationType.SEQUENCE
  - GenerationType.TABLE
- generates the primary key values programmatically
- use database features, like auto-incremented columns or sequences.
- We can use @GeneratedValue annotation to your primary key attribute for choosing the generation strategy.

Mapping Collections to Persistent class using Hibernate

- Collection elements can be mapped to persistent class in Hibernate
- Collection element has to be declared before using them.
- Collection object can be mapped to the persistent class either declaring them in mapping file or by annotation.

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

//getters and setters

}
```

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

    ...
</class>
```

QUIZ QUESTION

Which of the following is a benefit provided by ORM

- ❑ Automatic table generation
- ❑ Easy replacement of Database
- ❑ Persistence management in Object Oriented way.
- ❑ All of the above

QUIZ QUESTION

Which of the following object provides an interface between the Hibernate application and the data stored in the database?

- ❑ Transaction
- ❑ Session
- ❑ Configuration
- ❑ SessionFactory

SUMMARY

ORM with Hibernate

SUMMARY

In this lesson, you've learned to:

- Define Object Persistence
- Describe Limitations of JDBC
- Discuss ORM benefits
- Discuss Object Relational Mismatch
- Hibernate as ORM solution provider
- Understand Hibernate Advantages
- Map hibernate persistent classes
- Understand different keys