Hibernate

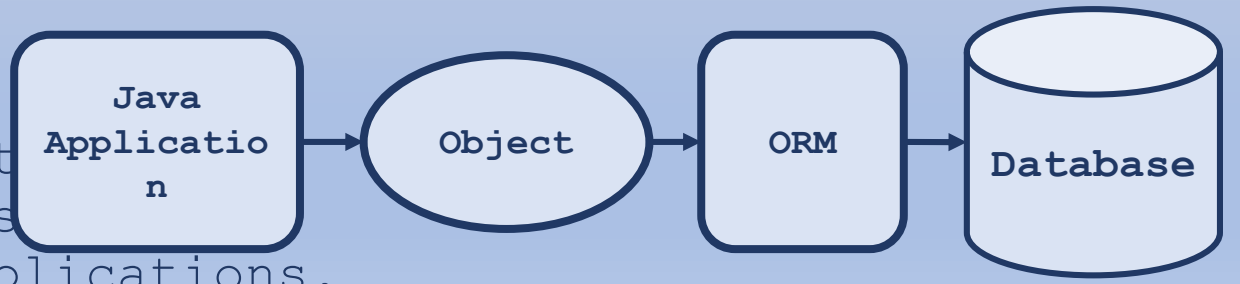
Unit - 2

Topics to be Covered...

- ORM
- JDBC V/s Hibernate
- Introduction to Hibernate
- Hibernate Architecture
- Setting Up Hibernate
- Exploring HQL
- Understanding O/R Mapping
- Working with O/R Mapping
  - Developing Hibernate Configuration File
  - Hibernate Mapping File
  - Java Beans

ORM: Object – Relational Model

- ORM is a programming technique for converting data between incompatible systems, specifically between object-oriented programming languages and relational databases.
- Simplifies data manipulation by abstracting database interactions into higher-level object-oriented constructs.
- Its benefits are:
  - **Productivity:** Reduces boilerplate code and speeds up development.
  - **Maintainability:** Promotes cleaner, more maintainable code by separating business logic from data access logic.
  - **Portability:** Makes it easier to switch between different database systems.
  - **Type Safety:** Ensures compile-time checking of data access code.
- Popular ORM Frameworks:
  - **Hibernate:** For Java applications.
  - **Entity Framework:** For .NET applicat
  - **SQLAlchemy:** For Python applications
  - **ActiveRecord:** For Ruby on Rails applications.
  - **Django ORM:** For Django applications in Python



JDBC v/s Hibernate

Feature	JDBC	Hibernate
Mapping Java Classes to Database Tables	Manual mapping required	Automatic mapping by Hibernate
Query Generation	Developers write SQL queries	Hibernate generates queries
Portability	Requires specific JDBC driver for databases	More portable, works with various databases
Complexity	Complex, developers manage mapping and SQL	Simplified, Hibernate handles mapping
SQL Support	Supports native SQL queries	Provides HQL (Hibernate Query Language)
Lazy Coding	No Supported by default	Allows loading of Objects at run-time and on-demand.
	Requires writing and	Reduces boilerplate coding

Hibernate Introduction

- Hibernate is the most used Object/Relational persistence and query service and It is licensed under the open-source GNU Lesser General Public License (LGPL).
- It is Open-sourced.
- It simplifies database interactions by mapping Java classes to database tables.
- Also provides data query and recovery facilities.
- Commonly used in enterprise applications for managing complex database interactions.
- Suitable for applications requiring robust data persistence, transaction management, and efficient data retrieval.

Hibernate Introduction

- It has features like,
  - **ORM:** Automatically maps Java objects to database tables.
  - **HQL:** Hibernate Query Language, similar to SQL but operates on Hibernate objects.
  - **Fast performance:** The performance of hibernate framework is fast because cache is internally used in hibernate framework.
  - **Caching:** Supports both first-level (session) and second-level (session factory) caching.
  - **Lazy Loading:** Fetches data only when it's actually needed, improving performance.
  - **Automatic Table Generation:** Can automatically generate database tables based on Java class definitions.
  - **Simplifies complex join:** To fetch data from multiple tables is easy in Hibernate framework.

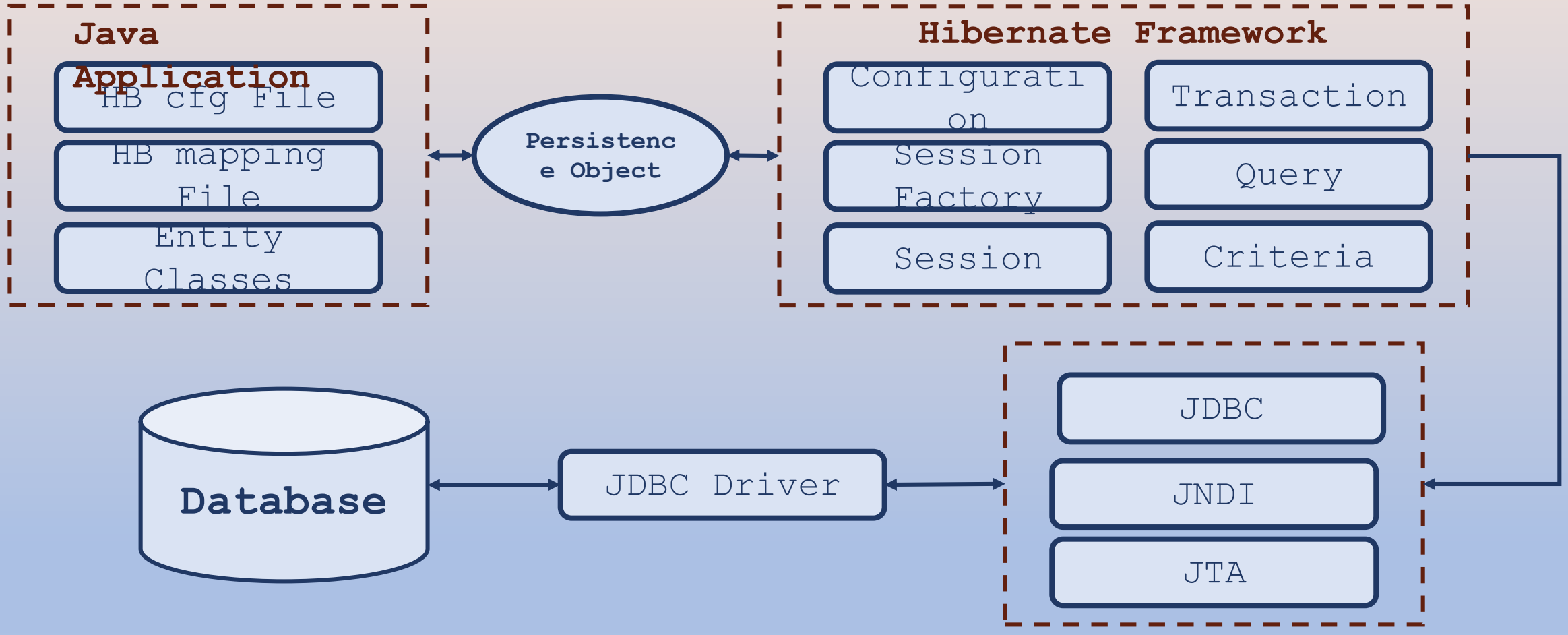
Hibernate Introduction

- Its benefits include,
  - **Simplified Data Access:** Reduces boilerplate code for database operations.
  - **Portability:** Abstracts away database-specific details, making it easier to switch databases.
  - **Transaction Management:** Provides integrated transaction management, making it easier to manage database transactions.
  - **Performance Optimization:** Supports batch processing, caching, and lazy loading to enhance performance

Hibernate Introduction

- Typically configured using XML files (**hibernate.cfg.xml**) or annotations within Java classes.
- Supports **various configurations** for **database connections, caching, and query optimization.**
- Uses **Java annotations** (e.g., **@Entity, @Table, @Id etc.**) to define mappings between Java classes and database tables
- Manages database operations through **Session** objects, which are lightweight and designed for short-lived use.
- Provides methods for CRUD operations (**save, update, delete, get, load**)
- Java Persistence API (JPA) is a Java specification that provides specific functionality and is a standard for ORM tools.
- **javax.persistence** package contains the JPA classes and interfaces.

Hibernate Architecture

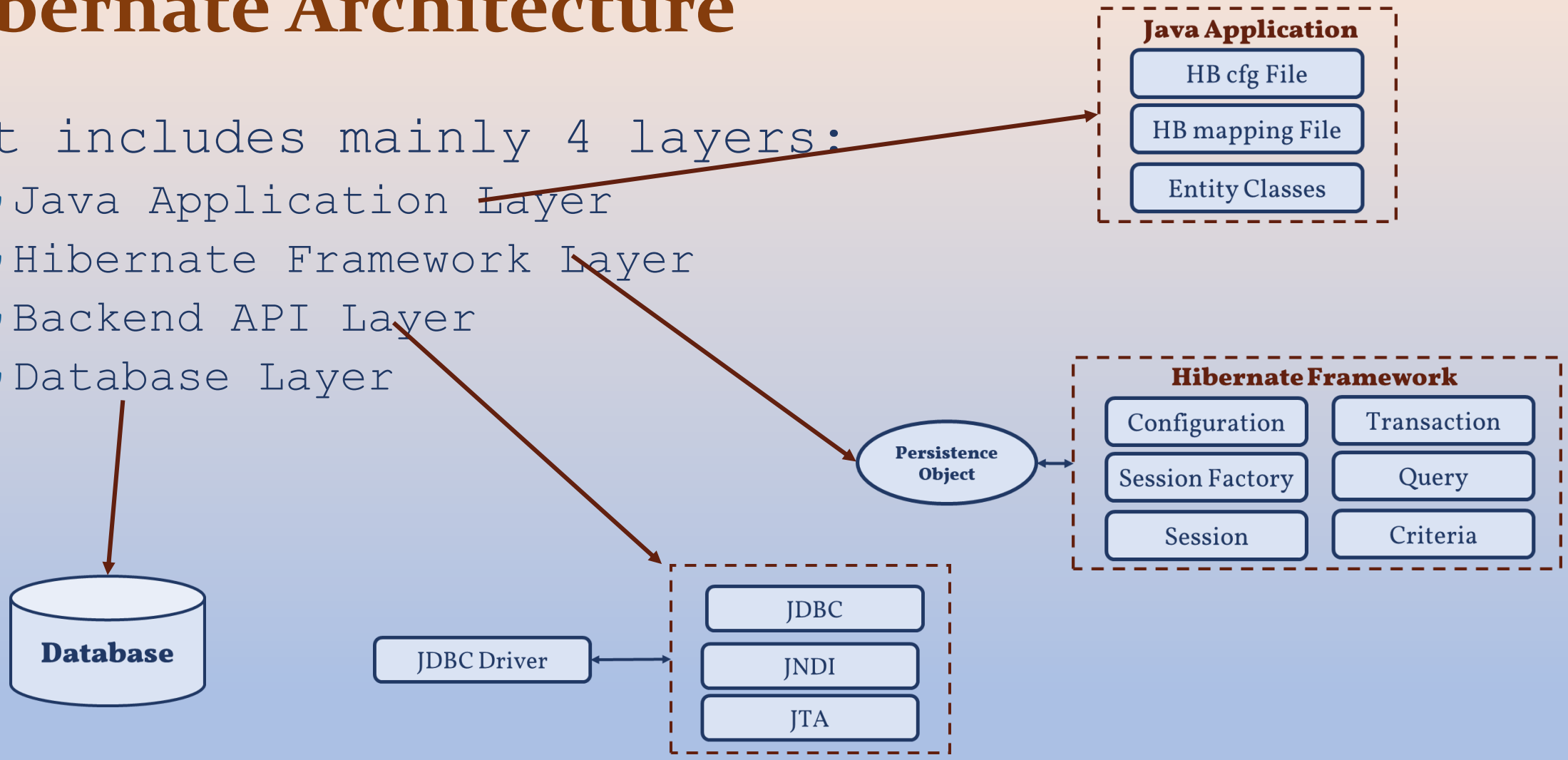


#JNDI: *Java Naming & Directory Interface*

#JTA: *Java Transaction API*

Hibernate Architecture

- It includes mainly 4 layers:
  - Java Application Layer
  - Hibernate Framework Layer
  - Backend API Layer
  - Database Layer



Hibernate Architecture

- Hibernate uses persistence, what is that?
- Persistence simply means that we would like our application's data to outlive the applications process. In Java terms, 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.

Hibernate Architecture

- Internal API used by Hibernate:
  - **JDBC (Java Database Connectivity):**
    - A standard Java API for connecting and executing queries against databases. (Already discussed in Unit-I)
    - Provides methods for querying and updating data in a relational database.
    - Supports executing SQL statements, managing connections, and processing results.
  - **JTA (Java Transaction API):**
    - A Java API for managing and coordinating transactions across multiple resources.
    - Allows applications to perform distributed transactions, ensuring all involved resources commit or rollback together.
    - Provides interfaces for demarcating transactions and managing transaction lifecycle.
  - **JNDI (Java Naming Directory Interface):**
    - A Java API for directory service and naming operations.
    - Allows Java applications to access various naming and directory services (e.g., LDAP, DNS).

Hibernate Architecture

- There are various key elements in Hibernate Architecture.
- **Configuration:**
  - It is the first required object for any Hibernate application.
  - Usually created during application initialization.
  - This object provides 2 key components:
    - **Database Connection:** Handled through one or more hibernate supported config files **(hibernate.properties, hiebernate.cfg.xml etc.)**
    - **Class Mapping Setup:** Component responsible for creating connection between Java Classes and Database Tables.

Hibernate Architecture

- **Configuration:**
  - Configuration is a class which is present in `org.hibernate.cfg` package. It **activates Hibernate framework.** It **reads both configuration file and mapping files.**

```
Configuration cfg = new Configuration(); //Activates Hibernate
cfg.configure(); //reads both cfg file and mapping file.
```

  - It checks whether the config file is syntactically correct or not.
  - If the config file is not valid then it will throw an exception. If it is valid then it creates a meta-data in memory and returns the meta-data to object to represent the config file.

Hibernate Architecture

- **SessionFactory:**
  - SessionFactory is an Interface which is present in org.hibernate package and it is **used to create Session Object.**
  - It is **immutable and thread-safe** in nature.
  - You would need one SessionFactory object per database using a separate configuration file.
  - **buildSessionFactory():** method gathers the meta-data which is in the cfg Object.
  - From cfg object it takes the JDBC information and create a JDBC Connection.

Hibernate Architecture

- **Session:**
  - Session is an **interface** which is present in org.hibernate package. Session object is created **based upon SessionFactory object** i.e. factory.
  - It opens the Connection/Session with Database software through Hibernate Framework.
  - It is a light-weight object and it is not thread-safe.
  - Session object is used to perform CRUD operations.
  - Instantiated each time an interaction is needed with the database.
  - The session objects should not be kept open for a long time because they are not usually thread safe and they should be created and destroyed as needed.

```
Session session = factory.buildSession();
```

Hibernate Architecture

- **Transaction:**
  - Transaction object is used whenever we perform any operation and based upon that operation there is some change in database.
  - It represents a unit of work with the database.
  - Transactions in Hibernate are handled by an underlying transaction manager and transaction (From JDBC or JTA).
  - Transaction object is used to give the instruction to the database to make the changes that happen because of operation as a permanent by using commit() method.

```
Transaction tx = session.beginTransaction();
tx.commit();
```

Hibernate Architecture

- **Query:**
  - Query is an interface that present inside org.hibernate package.
  - Query objects use SQL or Hibernate Query Language (HQL) string to store/retrieve data to/from the database and create objects.
  - A Query instance is **obtained by** calling Session.createQuery().
  - A Query instance is used to bind query parameters, limit the number of results returned by the query, and finally to execute the query.
  - This interface exposes some extra functionality beyond that provided by Session.iterate() and Session.find():
    - A **particular page of the result set may be selected** by calling setMaxResults(), setFirstResult().
    - Named query parameters may be used.

Hibernate Architecture

- **Criteria:**
  - Criteria is a simplified API for retrieving entities by composing **Criterion** objects.
  - The Session is a factory for Criteria. Criterion instances are usually obtained via the factory methods on Restrictions.
  - Criteria objects are used to create and execute object oriented criteria queries to retrieve objects.

Setting-Up Hibernate Environment

- Download and Install Eclipse:
  - Download Eclipse IDE: https://www.eclipse.org/
  - Install Eclipse IDE using "Eclipse IDE for Java Developers" option.
- Download Hibernate Repository (.zip):

  https://sourceforge.net/projects/hibernate/files/
  - Extract the .zip into a "hibernate" directory.
- Create Hibernate Application:
  - Create a new Project: File -> New -> Project -> Java Project, give name and Finish.
  - Add .jar files: Right click on Project -> Build Path -> Add External Archives
  - In the new dialog, Select required .jar files

Hibernate Mapping Types

- While preparing a Hibernate mapping document, we map the Java data types into RDBMS data types.
- The types declared and used in the mapping files are not Java data types; they are not SQL database types either.
- These types are called ***Hibernate mapping types***, which can translate from Java to SQL data types and vice versa.

Hibernate Mapping Types:

Primitive Types

Mapping type	Java type	SQL Type
integer	int or java.lang.Integer	INTEGER
long	long or java.lang.Long	BIGINT
short	short or java.lang.Short	SMALLINT
float	float or java.lang.Float	FLOAT
double	double or java.lang.Double	DOUBLE
character	java.lang.String	CHAR(1)
byte	byte	TINYINT
boolean	boolean	BIT
true/false	boolean	CHAR(1) ('T' or 'F')

Hibernate O/R Mapping

Three most important mapping are as follows:

1. Collections Mappings
2. Association Mappings
3. Component Mappings

Hibernate O/R Mapping

Collections Mappings

- If an entity or class has collection of values for a particular variable, then we can map those values using any one of the collection interfaces available in java.
- Hibernate can persist instances of **java.util.Map, java.util.Set, java.util.SortedMap, java.util.SortedSet, java.util.List**, and any **array** of persistent entities or values.

Hibernate O/R Mapping:

Collection type	Mapping and Description
java.util.Set	This is mapped with a **<set>** element and initialized with java.util.HashSet
java.util.SortedSet	This is mapped with a **<set>** element. The sort attribute can be set to either a comparator or natural ordering.
java.util.List	This is mapped with a <**list**> element and initialized with java.util.ArrayList
java.util.Collection	This is mapped with a <**bag**> or <**ibag**> element and initialized with java.util.ArrayList
java.util.Map	This is mapped with a <**map**> element and initialized with java.util.HashMap
java.util.SortedMap	This is mapped with a <**map**> element. The sort attribute can be set to either a comparator or natural ordering.

Hibernate O/R Mapping:

Association Mappings:

- The mapping of associations between entity classes and the relationships between tables is the soul of ORM.
- There are the four ways in which the cardinality of the relationship between the objects can be expressed.
- An association mapping can be unidirectional as well as bidirectional.

Hibernate O/R Mapping:

Association Mappings:

Mapping type	Description
Many-to-One	Mapping many-to-one relationship using Hibernate
One-to-One	Mapping one-to-one relationship using Hibernate
One-to-Many	Mapping one-to-many relationship using Hibernate
Many-to-Many	Mapping many-to-many relationship using Hibernate

Hibernate O/R Mapping:

Component Mappings:

- If the referred class does not have it's own life cycle and completely depends on the life cycle of the owning entity class, then the referred class is called as the Component class.
- The mapping of Collection of Components is also possible in a similar way just as the mapping of regular Collections with minor configuration differences.

Mapping type	Description
Component Mappings	Mapping for a class having a reference to another class as a member variable.

Hibernate Query Language (HQL)

- The Hibernate ORM framework provides its own query language called Hibernate Query Language .
- Hibernate Query Language (HQL) is same as SQL (Structured Query Language) but it doesn't depends on the table of the database. Instead of table name, we use class name in HQL.

  Therefore, it is database independent query language.

Hibernate Query Language (HQL)

Characteristics of HQL

1. **Similar to SQL**

   HQL's syntax is very similar to standard SQL. If you are familiar with SQL then writing HQL would be pretty easy.

2. **Fully object-oriented:** HQL doesn't use real names of table and columns. It uses class and property names instead. HQL can understand inheritance, polymorphism and association.

3. **Reduces the size of queries**

HQL vs SQL

SELECT QUERY

SQL

ResultSet rs=st.executeQuery("**select * from diet**");

HQL

Query query= session.createQuery("**from diet**");

//here persistent class name is diet

HQL vs SQL

SELECT with WHERE clause

SQL

ResultSet rs=st.executeQuery("**select * from diet where id=301**");

HQL

Query query= session.createQuery("**from diet where id=301** ");

//here persistent class name is diet

HQL vs SQL

UPDATE QUERY

SQL

```
1.  String query = "update User set name=? where id = ?";
2.  PreparedStatement preparedStmt = conn.prepareStatement(query);
3.  preparedStmt.setString (1, “DIET_CE”);
4.  preparedStmt.setInt(2, 054);
5.  preparedStmt.executeUpdate();
```

HQL

```
1.  Query q=session.createQuery("update User set name=:n where id=:i");
2.  q.setParameter("n", "DIET_CE");
3.  q.setParameter("i",054);
4.  int status=q.executeUpdate();
```

HQL vs SQL

INSERT QUERY

SQL

String sql = "**INSERT INTO Stock VALUES (100, 'abc')**";

int result = stmt.executeUpdate(sql);

HQL

Query query = session.createQuery("**insert into Stock(stock_code, stock_name) select stock_code, stock_name from backup_stock**");

int result = query.executeUpdate();

Steps to run hibernate example

Steps to run first hibernate example with MySQL in Netbeans IDE 8.2

Step-1: Create the database

```
CREATE DATABASE retailer;
```

Step-2: Create table result

```
CREATE TABLE customers(
    name varchar(20),
    C_ID int NOT NULL AUTO_INCREMENT,
    address  varchar(20),
    email varchar(50),
    PRIMARY KEY(C_ID)
);
```

Steps to run hibernate example

Step-3: Create new java application.

- File > New project > Java > Java Application > Next Name it as **HibernateTest.**
- Then click Finish to create the project.

Steps to run hibernate example

Step-4: Create a POJO(Plain Old Java Objects) class

- We create this class to use variables to map with the database columns.
- Right click the package (hibernatetest) & select New > Java Class Name it as Customer.
- Click Finish to create the class.

Steps to run hibernate example: Step-4

```
1.  package hibernatetest;
2.  public class Customer {
3.    public String customerName;
4.    public int customerID;
5.    public String customerAddress;
6.    public String customerEmail;
7.  public void setCustomerAddress(String
                          customerAddress) {
      this.customerAddress = customerAddress;}
8.  public void setCustomerEmail(String customerEmail) {
9.           this.customerEmail = customerEmail;}
10. public void setCustomerID(int customerID) {
11.          this.customerID = customerID; }
```

Steps to run hibernate example: Step-4

```
12. public void setCustomerName(String customerName) {
13.         this.customerName = customerName;    }
14. public String getCustomerAddress() {
15.         return customerAddress;    }
16. public String getCustomerEmail() {
17.         return customerEmail;    }
18. public int getCustomerID() {
19.         return customerID;    }
20. public String getCustomerName() {
21.         return customerName;    }
22.}
```

Steps to run hibernate example: Step-4

Step-4: Create a POJO(Plain Old Java Objects) class

- To generate getters and setters easily in NetBeans, right click on the code and select Insert Code Then choose Getter... or Setter...
- Variable **customerName** will map with the name column of the **customers** table.
- Variable **customerID** will map with the **C_ID** column of the customers table. It is integer & auto incremented. So POJO class variable also should be int.
- Variable **customerAddress** will map with the **address** column of the customers table.
- Variable **customerEmail** will map with the **email** column of the customers table.

Steps to run hibernate example

Step-5: Connect to the database we have already created. [retailer]

- Select Services tab lying next to the Projects tab.
- Expand Databases.
- Expand MySQL Server. There we can see the all databases on MySQL sever
- Right click the database retailer. Select Connect.

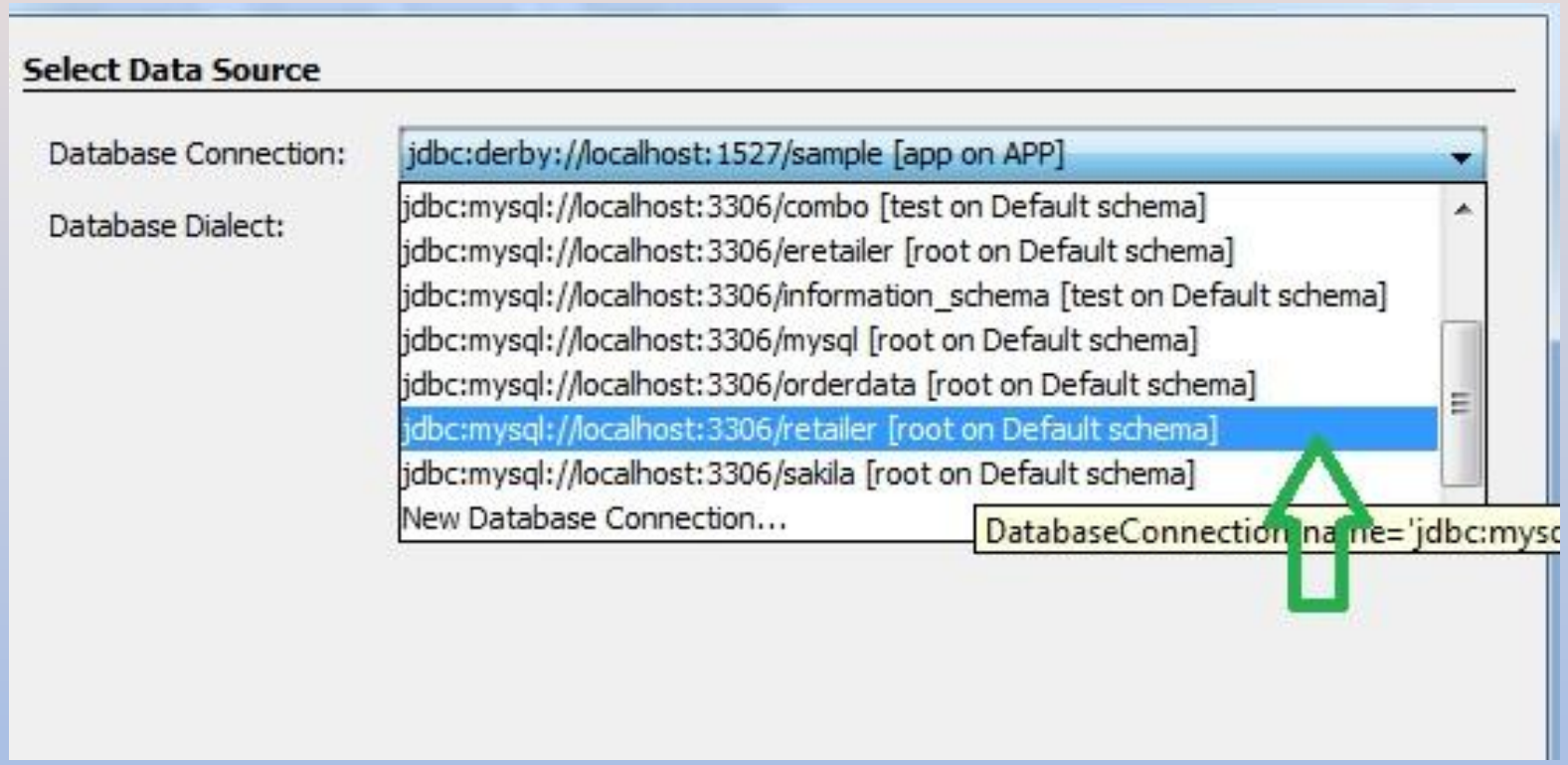
Steps to run hibernate example

Step-6: Creating the configuration XML

- Hibernate need a configuration file to create the connection.
- Right click package hibernatetest select New > Other > **Hibernate > Hibernate Configuration Wizard**
- Click Next >
- In next window click the drop down menu of Database Connection and select retailer database connection.

Steps to run hibernate example: Step-6

Step-6: Creating the configuration XML



- Click Finish to create the file.

Steps to run hibernate example

hibernate.cfg.xml

```
1. <hibernate-configuration>
2.   <session-factory>
3.     <property name="hibernate.connection.driver_class">
                        com.mysql.jdbc.Driver </property>
4.     <property name="hibernate.connection.url">
   jdbc:mysql://localhost:3306/retailer</property>
5.     <property name="hibernate.connection.username">
                              root</property>
6.     <property name="hibernate.connection.password">
                              root</property>
7.     <property name="hibernate.connection.pool_size">
                                10</property>
8.     <property name="hibernate.dialect">
   org.hibernate.dialect.MySQLDialect</property>
```

Steps to run hibernate example

```
9.  <property name="current_session_context_class">
                                    thread</property>
10. <property name="cache.provider_class">
        org.hibernate.cache.NoCacheProvider</property>
11. <property name="show_sql">true</property>
12. <property name="hibernate.hbm2ddl.auto">
                                  update</property>
13. <mapping resource="hibernate.hbm.xml"></mapping>
14. </session-factory>
15. </hibernate-configuration>
```

Steps to run hibernate example

Step-7: Creating the mapping file [hibernate.hbm]

- Mapping file will map relevant java object with relevant database table column.
- Right click project select New > Other > Hibernate **> Hibernate Mapping Wizard**
- click Next name it as hibernate.hbm
- click Next> In next window we have to select Class to Map and Database Table.
- After selecting correct class click OK
  Select Database Table
  Click drop down list and select the table you want to map.
  Code for mapping file.

Steps to run hibernate example: Step-7

```
1.  <hibernate-mapping>
2.    <class name="hibernatetest.Customer" table="customers">
3.    <id column="C_ID" name="customerID" type="int">
4.    <generator class="native">
5.    </generator></id>
6.    <property name="customerName">
7.    <column name="name">
8.    </column></property>
9.    <property name="customerAddress">
10.   <column name="address">
11.   </column></property>
12.   <property name="customerEmail">
13.   <column name="email">
14.   </column></property>
15.   </class></hibernate-mapping>
```

hibernate.hbm.xml

Steps to run hibernate example: Step-7

Step-7: Creating the mapping file [hibernate.hbm]

- property name = variable name of the POJO class
- column name = database column that maps with previous variable

Steps to run hibernate example

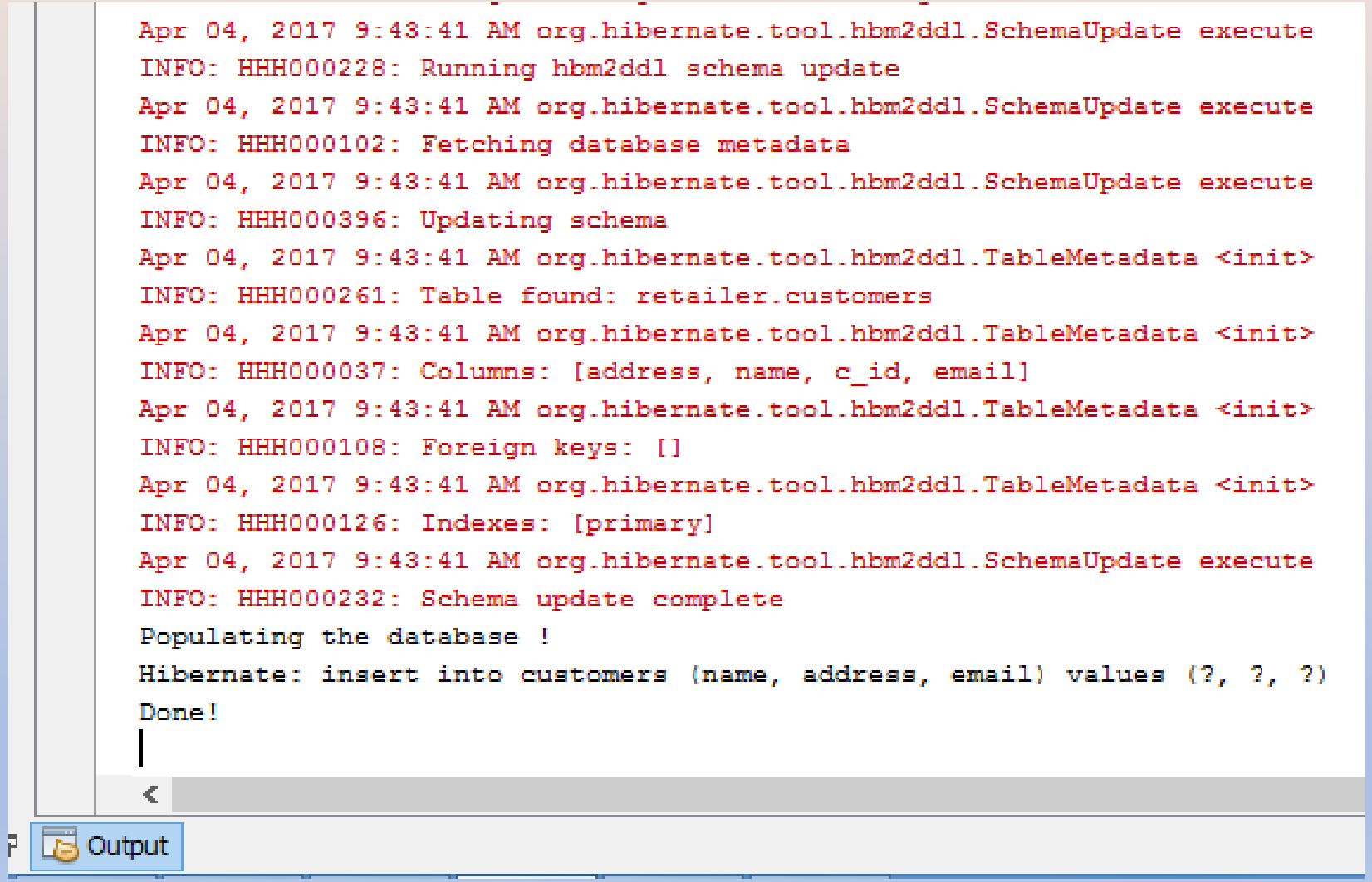
Step-8: Now java program to insert record into the database

```
1.  package hibernatetest;
2.  import org.hibernate.Session;
3.  import org.hibernate.SessionFactory;
4.  public class HibernateTest {
5.       public static void main(String[] args) {
6.          Session session = null;
7.  try
8.  {
9.  SessionFactory sessionFactory = new
    org.hibernate.cfg.Configuration().configure().buildSessionFactory()
    ;
```

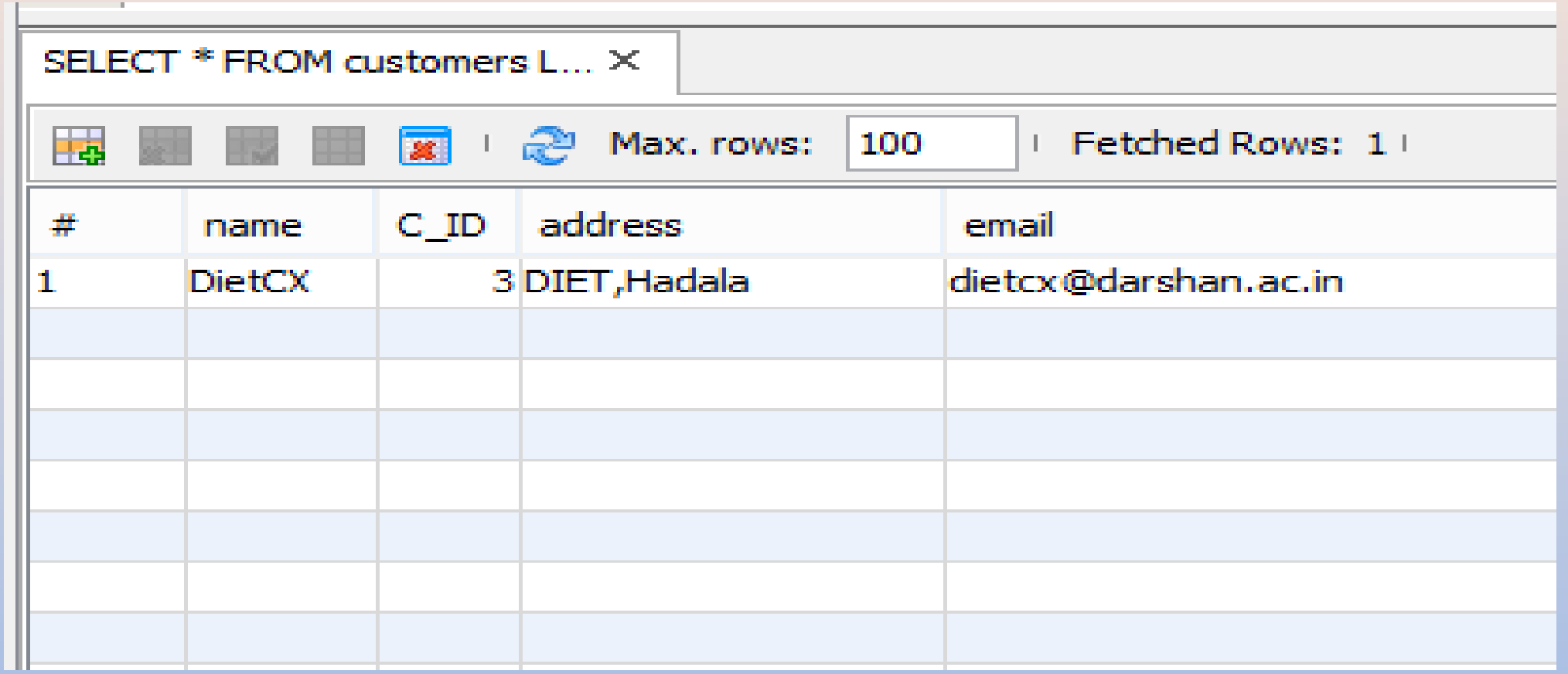
Steps to run hibernate example

```
10. session =sessionFactory.openSession();
11.         session.beginTransaction();
12.         System.out.println("Populating the database !");
13.         Customer customer = new Customer();
14.         customer.setCustomerName("DietCX");
15.         customer.setCustomerAddress("DIET,Hadala");
16.         customer.setCustomerEmail("dietcx@darshan.ac.in");
17.         session.save(customer);
18.         session.getTransaction().commit();
19.         System.out.println("Done!");
20.         session.flush();
21.         session.close();
22.       }catch(Exception e)       {System.out.println(e.getMessage());
   } } }
```

Steps to run hibernate example:output



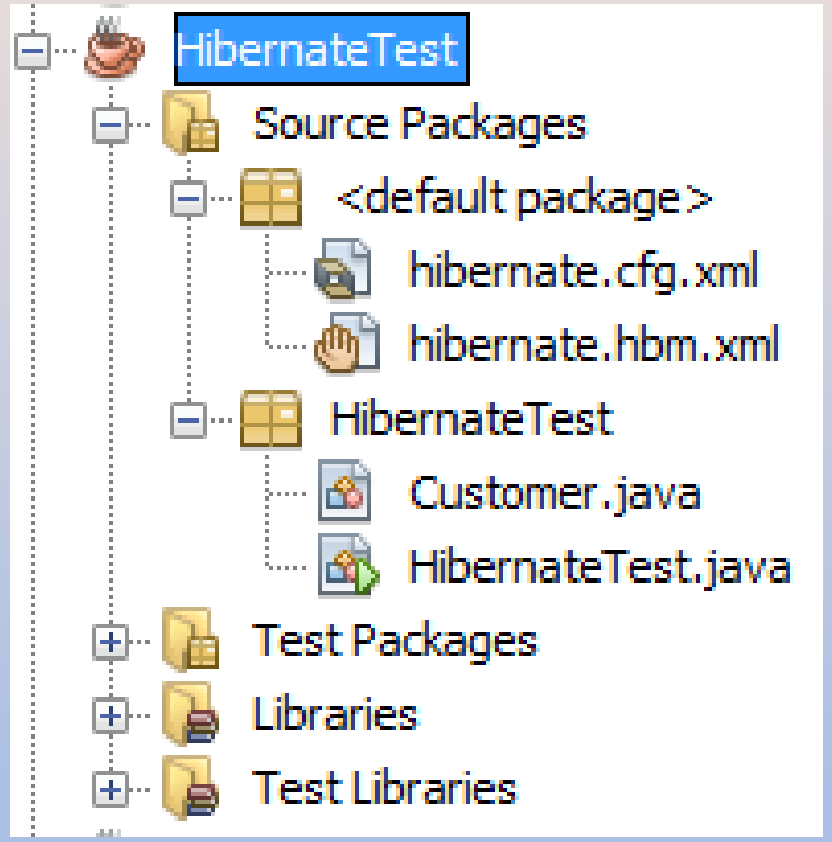
Steps to run hibernate example: output


SELECT * FROM customers L...

Max. rows: 100 Fetched Rows: 1

#	name	C_ID	address	email
1	DietCX	3	DIET,Hadala	dietcx@darshan.ac.in

Hibernate Program Hierarchy



GTU Questions

1	Explain the Hibernate cache architecture.
2	What is HQL? How does it different from SQL? List its advantages.
3	What is OR mapping? Give an example of Hibernate XML mapping file.
4	What is HQL? How does it differ from SQL? Give its advantages.
5	Draw and explain the architecture of Hibernate.
6	Explain architecture of Hibernate.
7	Explain architecture of Spring MVC Framework. Explain all modules in brief.
8	What is O/R Mapping? How it is implemented using Hibernate. Explain with example.
9	What are the advantages of Hibernate over JDBC?
10	What is hibernate? List the advantages of hibernate over JDBC.
11	Develop program to get all students data from database using hibernate. Write necessary xml files.