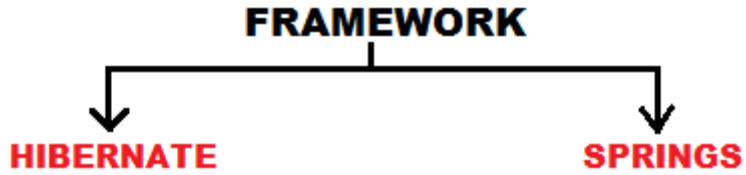
❖ INTRODUCTION



[Advanced J2EE] **FRAMEWORK** ⇒ 1) HIBERNATE ⇒ JDBC
⇒ 2) SPRINGS ⇒ SERVLETS

Why the other name for Framework?

- Because in case of FRAMEWORK we can perform much more additional functionalities compared to J2EE.

How are we able to perform functionalities in case of Framework?

- Because additional set of libraries present in framework in the form of JAR files.

- Maximum JAR files we use in J2EE [2-3].
- The total numbers of JAR files used in single application in case of FRAMEWORK are [13-15].
- Problem with JDBC is Duplication of Code.

✸ **JDBC problem:-**

✸ **SAVE:**

1. Load and register the Driver [Driver classes].
2. Establish a connection with the Database Server.
3. Create a statement or platform.
4. Execute the SQL statement or SQL queries.
5. Close all the costly resources.

✸ **UPDATE:**

1. Load and register the Driver [Driver classes].
2. Establish a connection with the Database Server.
3. Create a statement or platform.
4. Execute the SQL statement or SQL queries.
5. Close all the costly resources.

✸ **DELETE:**

1. Load and register the Driver [Driver classes].
2. Establish a connection with the Database Server.
3. Create a statement or platform.
4. Execute the SQL statement or SQL queries.
5. Close all the costly resources.

- In all the cases, all the steps will remain the same except the 4th step; this condition is called **Duplication of code** (or) **Boiler Plate Code.**
- To avoid this **"Duplication of code** (or) **Boiler Plate Code"**, we use **SingleTon Class.**
- Where **SingleTon** class always returns 2 methods

a) for Connection ⇒ Trying to establish a connection with the DB server

b) closing the Connection ⇒ To close the Connection [since it is a costly resource]

✸ **JDBC is always dependent on Database.**

[Since the SQL queries varies from one Database Server to another.]

- Hibernate is independent of Database, since hibernate automatically generates SQL queries as per the Database

✸ **JDBC does not support auto-generation of Tables & Primary Key.**

- In case of hibernate, hibernate will automatically generate the tables on its own along with assigning primary key.

✸ **JDBC does not support "cache mechanism".**

"**Cache memory**":- It is a temporary (or) buffer area, which stores constant data / repeated data from the Database.

(**) cache mechanism avoids traffic between Java Application and Database Server, hence performance of an application increases.

2. JAVA problem:-

```
public class Student {
      private int id;
      private String name;
     //Getters and Setters...
}
```

Java Bean Class (or)
Entity Class (or)
POJO Class
[Plain Old Java Object]

```
public class Test {
      public static void main(String[] args) {
            Student s=new Student();
            s.setId(24);
            s.setname("Test");
      }
}
```

SQL Language + Java Language(Object)

String qry="insert into btm.student values(s);

Error

- In case of Java, once we create object and then if we directly pass object as value in the query, then there is a problem. The query will not accept instead it throws an error.

In case of Java, an Object holds 3 things

1. Identity :- Unique ID of the Object (hash code)
2. State :- Data/value of the Object
3. Behaviour :- Functionality (or) methods

3. Inheritance Problem:- [Is-A relationship]

```
public class User {
      private int id;
      private String name;
      private String address;
     //Getters and Setters...
}
```

User

Id	Name	Address
1	Uday	Jspiders

String qry="insert into btm.User values(1,'Uday','Jspiders');

```
public class Trainer extends User {
      private double salary;
     //Getters and Setters...
}
```

Trainer

salary
20000.00

String qry="insert into btm.Trainer values(20000.00);

- In case of JDBC, multiple queries have to be executed to save the data into the multiple tables.

- In case of Hibernate, hibernate will automatically generate one single query on its own and the data's are automatically saved into different tables at once.

> **Note: (***)**
> - In case of Hibernate, we never deal with normal class.
> - We always use only Java Bean Class / Entity Class / POJO Class, because Hibernate works on a concept called **ORM**(**O**bject **R**elational **M**odel)

4. Composition Problem: [Has-A relationship]

```
public class Student {
      private int id;
      private String name;
      private List<Course> course;
     //Getters and Setters...
}
```

Student

id	name	Cid
1	Uday	2
2	Kiran	1

```
public class Course {
      private int Cid;
      private String Cname;
     //Getters and Setters...
}
```

Course

Cid	Cname
1	Java
2	J2EE

- In case of JDBC, individual queries are needed to affect multiple tables.
- In case of Has-A relationship, hibernate will use annotations called @OneToMany etc., and automatically data's will be saved into two tables.

❖ ORM [Object Relational Mapping]:-

```
public class Student {          ⇒ Class
      public int id;
      public String name;       ⇒ Variables
      public double perc;

public static void main(String arg[]){
      Student s=new Student();
      s.id=1;
      s.name="Uday";            ⇒ Object
      s.perc=69.2;
  }
}
```

Object Model

Table ⇐ **Student**

Columns ⇐

Record/row ⇐

id	name	perc
1	Uday	69.2

Data Model Or Relational Model

ORM

Class	=	Table
Variables	=	Columns
Object	=	Record/row

Object Model

Why it is called as Object Model?

Until we create an Object we cannot access one single functionality

Data Model (or) Relational Model

Why it is called as Data Model?

Apart from data we don't find any other components

"Conversion of Java Object into Relational Model is known as Object Relational Mapping".

Or

"The Object which created in case of Java must mandatorily have a relationship with data's in the Database server is known as **Object Relational Mapping** [**ORM**]".

- There are different types of mechanism / tools present in ORM namely;
  i. Hibernate
  ii. TopLink
  iii. Ibatis etc...

❖ **Specifications of ORM:-**

There are 3 different specifications wrt ORM;

i. Every Java Bean Class / Entity Class / POJO class represents a Table.
ii. Every Variable of Java Bean Class represents a Column.
iii. Every Java Bean Class Object represents a Record/row.

ORM

Class	=	Table
Variables	=	Columns
Object	=	Record/row

❖ FRAMEWORK:-

- Framework is a set of API's (or) semi-software (pre-written code) which is used to develop an application in a loosely coupled manner.
  Ex.:- Ready-made template
- It is used to develop an application in a simplified manner (or) using which we can develop an application in a simplified manner.
- There are 2 different categories present in framework;
  i. Invasive framework
  ii. Non-invasive framework

i. **Invasive Framework:-**

"***Framework which allows to*** extends one of their classes / implements one of their interfaces".

Ex.:- Struts, EJB framework

ii. **Non-invasive Framework:-**

"***Framework which does not allow to*** extend one of their classes / implements one of their interfaces".

Ex.:- Hibernate and Springs.

(***)Difference between API and Framework

:API:	:FRAMEWORK:
Is an interface between two different application	Is used to design / develop application such as MVC web application
Ex.: Java-Collection	Ex: Java-Array

❖ HIBERNATE:-

"Hibernate is an open-source, non-invasive framework, ORM tool which is used to convert Java Model [classes & variables] into Relational Model [Tables & Columns]"

❖ Advantages of Hibernate:

1. Hibernate is independent of Database.
   [Since SQL queries are automatically generated by hibernate as per the Database]
2. Hibernate supports auto-generation of Tables and Primary key.
3. Hibernate supports cache mechanism (avoids traffic between java application and Database server).
4. Hibernate supports Connection Pooling (resources which stores Database connection)
   [i.e., from one single java application we can hit multiple Database servers at a time]
5. Hibernate supports HQL [Hibernate Query Language]
6. (***) Hibernate supports dialect

Dialect: - **i)** It is responsible for generating SQL queries based on each Database

ii) To achieve Object Relational Mapping.

(***)Note:

1. **In case of Hibernate, all the checked exceptions are converted into unchecked exception which will be thrown at the run-time.**
2. **Hibernate can be used without server [Java application] as well as with server [JEE application]**

❖ Pre-requisites for Hibernate Java Application:-

1. Create a Java Bean Class / Entity Class / POJO Class.
2. Mapping between Java application and Relational Model which can be done in 3 different ways namely;
   a) xml file called as Filename.hbm.xml where **hbm** refers to Hibernate Mapping
   b) JPA Annotations in POJO class
3. Configuration file which is used to connect to the Database [**hibernate.cfg.xml]**
4. Test DAO class which "contains main method to execute the application".

DAO class ⇨ Data Access Object

Why we use DAO class?

- Since DAO class is the only area where we can access the properties or functionalities of Java Bean Class / Entity Class / POJO Class
- Only class / area to execute java application which has main method in it. [JVM is responsible for execution]

5. Add Hibernate jars into lib folder.
6. Add configuration file into the source folder [hibernate.cfg.xml]

❖ JPA Annotations: [must be declared only in Java Bean Class]

@Entity	⇨ specifies it is an Entity class which is used to write Hibernate logic
@Table	⇨ specifies Table name in the Database which maps Entity [Table name will be associated with Entity class]
@Id	⇨ specifies it is Primary Key where the data type must be int or long
@GeneratedValue	⇨ specifies auto-generation of Primary Key [because of this hibernate supports auto-generation of PK]

❖ Steps for execution of Hibernate Java Application:-

1. Open Java Perspective and select *Navigator* mode
2. Right click within Navigator mode and create a *new* Java Project & name it
3. Right click on Project and create a new folder called lib and add all the Jar files into the lib folder and build a java path to import the properties from the jar file.
4. Add the configuration file [hibernate.cfg.xml] into the source folder
5. Select src folder and create a new package structure
6. Select application name and create a class [java bean class / entity class / POJO class]

❖ Steps to be written in a DAO class:-

1. Create an object of **Java Bean Class** / **Entity class** / **POJO class**
2. Set the value of the members of Java Bean Class or Entity class
3. Create an object of Configuration class present in org.hibernate.cfg package
   ***Syntax*:**
   ```
   Configuration conf =new Configuration();
   ```
4. Call a zero-parameterised `configure()` method which is declared in Configuration Class.
   ***Syntax*:**
   ```
   conf =configure();
   ```

   Why to call a Zero-parameterised `configure()` ?

   `configure()` is responsible for loading and validating the configuration file called hibernate.cfg.xml

5. Create an implementation object of **SessionFactory** *interface* present in org.hibernate package by using a factory / helper method called *buildSessionFactory( )* which is declared inside Configuration class
   ***Syntax*:**
   ```
   SessionFactory sef=conf.buildSessionFactory();
   ```
6. Create an implementation object of **Session** *interface* present in org.hibernate package by using a factory / helper method called *openSession( )* which is declared inside **SessionFactory** *interface*
   ***Syntax*:**
   ```
   Session ses= sef.openSession();
   ```
7. Create an implementation object of **Transaction** *interface* present in org.hibernate package by using a factory / helper method called *beginTransaction( )* which is declared inside **Session** *interface*
   ***Syntax*:**
   ```
   Transaction tran= ses.beginTransaction();
   ```

8. Perform **CRUD** operation by using the reference of **Session**
   ***Syntax*:**
   ```
   ses.save(Object);
   ```
   `save(Object)` is declared in **Session** *interface* responsible to save / insert data's into the Database (only objects of java bean class)
9. Commit the **Transaction** in order **to save** the data into the Database server
   Syntax:
   ```
   tran.commit()
   ```
10. Close the Session
    Since the Session is considered to be a costly resource, so we need to close the costly resource
    Syntax:
    ```
    ses.close()
    ```

Program to insert / save the data:

hibernate.cfg.xml

```
<! DOCTYPE hibernate-configuration PUBLIC
      "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
      "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
      <session-factory>
         <!-- database information -->
         <property name="hibernate.connection.driver_class">
         com.mysql.jdbc.Driver</property>
         <property name="hibernate.connection.url">
          jdbc:mysql://localhost:3306/hibernate</property>
         <property name="hibernate.connection.user">
          root</property>
         <property name="hibernate.connection.password">
          root</property>
         <property name="dialect">
          org.hibernate.dialect.MySQL5Dialect</property>
         <property name="show_sql">true</property>
         <property name="hbm2ddl.auto">update</property>
         <mapping class="org.kuk.insertApp.Student"/>
      </session-factory>
  <hibernate-configuration>
```

Student.java

```
package org.kuk.insertApp;
import javax.persistence.*;
@Entity
@Table(name="Student")
public class Student {
      @Id
      @GeneratedValue
      private int id;
```

```
	private String name;
	private String address;
	public String getAddress() {
		return address;
	}
	public void setAddress(String address) {
		this.address = address;
	}
	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;
	}
}
```

TestDao.java

```
package org.kuk.insertApp;
import java.util.Scanner;
import org.hibernate.*;
import org.hibernate.cfg.*;
public class TestDao {
	public static void main(String[] args) {
		Scanner sc=new Scanner(System.in);
		System.out.println("Enter ur name: ");
		String name=sc.next();
		System.out.println("Enter ur address: ");
		String address=sc.next();
		Student stu=new Student( );
		stu.setName(name);
		stu.setAddress(address);
		Configuration confg=new Configuration();
		confg.configure();
		SessionFactory sef=confg.buildSessionFactory();
		Session ses=sef.openSession();
		Transaction tran=ses.beginTransaction();
		ses.save(stu);
		tran.commit();
		ses.close();
		sc.close();
	}
}
```

- Either to ***update*** / ***delete*** a record, we need to fetch the record first and then we can either ***update*** / ***delete*** a record by using 2 methods namely;
    1. **get( )**
    2. **load( )**
- **get( )** and **load( )** is used to fetch the record from th Database server based on a only one factor called ***Id.***
- **get( )** directly hits the Database server and returns the **real object** / **actual object.**
- **load( )** always returns a **Proxy object** without hitting the Database server.
- **Proxy object** is an Object with a given ***Id*** but the properties are not initialized yet.
- If the ***Id*** is not present in the table, then

  get( ) `throws NullPointerException` (or) `null`
- If the ***Id*** is not present in the table, then

  load( ) `throws ObjectNotFoundException`

Program to Update the data:

hibernate.cfg.xml

```
<!DOCTYPE hibernate-configuration PUBLIC
      "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
      "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
      <session-factory>
       <!-- database information -->
         <property name="hibernate.connection.driver_class">
          com.mysql.jdbc.Driver</property>
         <property name="hibernate.connection.url">
          jdbc:mysql://localhost:3306/hibernate</property>
         <property name="hibernate.connection.user">
          root</property>
         <property name="hibernate.connection.password">
          root</property>
         <property name="dialect">
          org.hibernate.dialect.MySQL5Dialect</property>
         <property name="show_sql">true</property>
         <property name="hbm2ddl.auto">update</property>
         <mapping class="org.kuk.updateApp.Student"/>
      </session-factory>
</hibernate-configuration>
```

Student.java

```
package org.kuk.updateApp;
import javax.persistence.*;
@Entity
@Table(name="Student")
```

```java
public class Student {
	@Id
	@GeneratedValue
	private int id;
	private String name;
	private String address;
	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 getAddress() {
		return address;
	}
	public void setAddress(String address) {
		this.address = address;
	}
  }
```

TestDao.java

```java
package org.kuk.updateApp;
import org.hibernate.*;
import org.hibernate.cfg.Configuration;
public class TestDao {
	public static void main(String[] args) {
		Configuration confg=new Configuration();
		confg.configure();
		SessionFactory sef=confg.buildSessionFactory();
		Session ses=sef.openSession();
		Transaction tran=ses.beginTransaction();
		Student student=(Student)ses.get(Student.class, 1);
		student.setName("UdayKiranK");
		student.setAddress("Karnataka");
		ses.update(student);
		tran.commit();
		ses.close();
	}
  }
```

Program to Delete the data:

hibernate.cfg.xml

```
<!DOCTYPE hibernate-configuration PUBLIC
      "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
      "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
      <session-factory>
        <!-- database information -->
         <property name="hibernate.connection.driver_class">
          com.mysql.jdbc.Driver</property>
         <property name="hibernate.connection.url">
          jdbc:mysql://localhost:3306/hibernate</property>
         <property name="hibernate.connection.user">
          root</property>
         <property name="hibernate.connection.password">
          root</property>
         <property name="dialect">
          org.hibernate.dialect.MySQL5Dialect</property>
         <property name="show_sql">true</property>
         <property name="hbm2ddl.auto">update</property>
         <mapping class="org.kuk.deleteApp.Student"/>
  </session-factory>
</hibernate-configuration>
```

Student.java

```
package org.kuk.deleteApp;
import javax.persistence.*;
@Entity
@Table(name="Student")
public class Student {
      @Id
      @GeneratedValue
      private int id;
      private String name;
      private String address;
      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 getAddress() {
		return address;
	}
	public void setAddress(String address) {
		this.address = address;
	}
}
```

TestDao.java

```
package org.kuk.deleteApp;
import org.hibernate.*;
import org.hibernate.cfg.Configuration;


public class TestDao {
	public static void main(String[] args) {
		Configuration confg=new Configuration();
		confg.configure();
		SessionFactory sef=confg.buildSessionFactory();
		Session ses=sef.openSession();
		Transaction tran=ses.beginTransaction();
		Student student=(Student)ses.get(Student.class, 1);
		ses.delete(student);
		tran.commit();
		ses.close();
	}
}
```

** if ***id*** is not present it **throws** java.lang.*IllegalArgumentException*

Note:

SQL queries ⇒ dialect

Primary Key ⇒ Generator

❖ Generator:-

- "**Generator** is the one which is used to generate a primary key value based on Generation strategy / Generation Algorithm".
- It is a responsible for auto-generation of Primary Key.
- There are 2 different categories of generator present;
  i. JPA generator
  ii. Hibernate generator

i. **JPA Generator:-** JPA generator supports 4 different types of Primary Key Generation Strategy which are as follows;

a. GenerationType.**AUTO**
b. GenerationType.**IDENTITY**
c. GenerationType.**SEQUENCE**
d. GenerationType.**TABLE**

a. GenerationType.**AUTO**:

- It is a ***default*** GenerationType which selects the generation strategy based on Database specific **dialect**
  [Based on the respective Database specifications, according to that primary key will be generated automatically]

Why default GenerationType**:-** For 2 important reasons;

1. Since it is supported by all the Database servers. Hence, the name ***default*** GenerationType
2. Whenever we **don't mention** the GenerationType, by ***default*** the GenerationType will be considered as GenerationType.**AUTO**

Syntax:

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

- Whenever we delete the data from actual table the record will not be deleted from comparison table
- Whenever we add new record on actual table the record will be added into comparison table

Comparison Table	**Actual Table**
1 ✓	1 ✓
2 ✓	2 ✓
3 ✓	3 ✗
4 ✓	4 ✓
	USER

b. GenerationType.**IDENTITY**:

- This GenerationType relies on an auto-incremented Database column

Syntax:

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

- It will check id present in Database and then will automatically increment the value based on that id in Database

c. GenerationType.**SEQUENCE**:

- This GenerationType is responsible for generating a primary key based on **sequence algorithm**
- This GenerationType is supported by only Oracle, IBM DB2, Postgres Database server

Syntax:

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

d. GenerationType.**TABLE**:

- This GenerationType is responsible for generating a primary key based on **Table algorithm**

Syntax:

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

```
1st value of id to be inserted id=1 for the 1st time
2nd time   : id= 32768
3rd time   : id= 65536
4th time   : id=98304
5th time   : id=131072
```

❖ Hibernate Generator:

- It supports many types of primary key Generation strategy which are as follows;
  a. Increment
  b. Foreign
  c. Identity
  d. Sequence
  e. Hilo
  f. Seqhilo
  g. Uuid
  h. Guid
  i. Assigned etc...

a. ***Increment***:-

- It generates the identifiers of type int, long or short that is unique where no other process is inserting the data into the same table
- Increment always auto-increments the value of Primary Key based on the maximum value of primary key present in the table
  [Maximum value of **Primary Key + 1**]

Syntax:

```
@Id
@GeneratedValue(generator="mygen")
@GenericGenerator(name="mygen",strategy= "increment")
```

hibernate.cfg.xml

```
<! DOCTYPE hibernate-configuration PUBLIC
      "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
      "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
      <session-factory>
       <!-- database information -->
       <property name="hibernate.connection.driver_class">
            com.mysql.jdbc.Driver</property>
       <property name="hibernate.connection.url">
            jdbc:mysql://localhost:3306/hibernate</property>
       <property name="hibernate.connection.user">
            root</property>
       <property name="hibernate.connection.password">
            root</property>
       <property name="dialect">
            org.hibernate.dialect.MySQL5Dialect</property>
       <property name="show_sql">true</property>
       <property name="hbm2ddl.auto">update</property>
       <mapping class="org.kuk.insertApp.Student"/>
  </session-factory>
</hibernate-configuration>
```

Student.java

```
package org.kuk.insertApp;
import javax.persistence.*;
@Entity
@Table(name="Student")
public class Student {
      @Id
      @GeneratedValue(generator="mygen")
      @GenericGenerator(name="mygen",strategy= "increment")
      private int id;
      private String name;
      private String address;
      public String getAddress() {
            return address;
      }
      public void setAddress(String address) {
            this.address = address;
      }
      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;
    }
}
```

TestDao.java

```
package org.kuk.insertApp;
import java.util.Scanner;
import org.hibernate.*;
import org.hibernate.cfg.*;
public class TestDao {
    public static void main(String[] args) {
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter ur name: ");
        String name=sc.next();
        System.out.println("Enter ur address: ");
        String address=sc.next();
        Student stu=new Student( );
        stu.setName(name);
        stu.setAddress(address);
        Configuration confg=new Configuration();
        confg.configure();
        SessionFactory sef=confg.buildSessionFactory();
        Session ses=sef.openSession();
        Transaction tran=ses.beginTransaction();
        ses.save(stu);
        tran.commit();
        ses.close();
        sc.close();
    }
}
```

b. ***Foreign***:-

- It uses the identifiers of another associated object which is generally used in conjunction with **<OneToOne>** Primary Key Association

❖ Cache Mechanism:

Cache memory: - it is temporary / buffer area which stores constant data or repeated data from the Database

Storage of cache memory: cache memory is generally stored in **RAM**

✷ Advantage of cache mechanism:-

- Cache mechanism avoids the traffic between Java application and Database server.
  Hence, the performance of an application increases.

There are 2 different types of cache present namely;

i. First-Level Cache
ii. Second-Level Cache

i. First-Level Cache

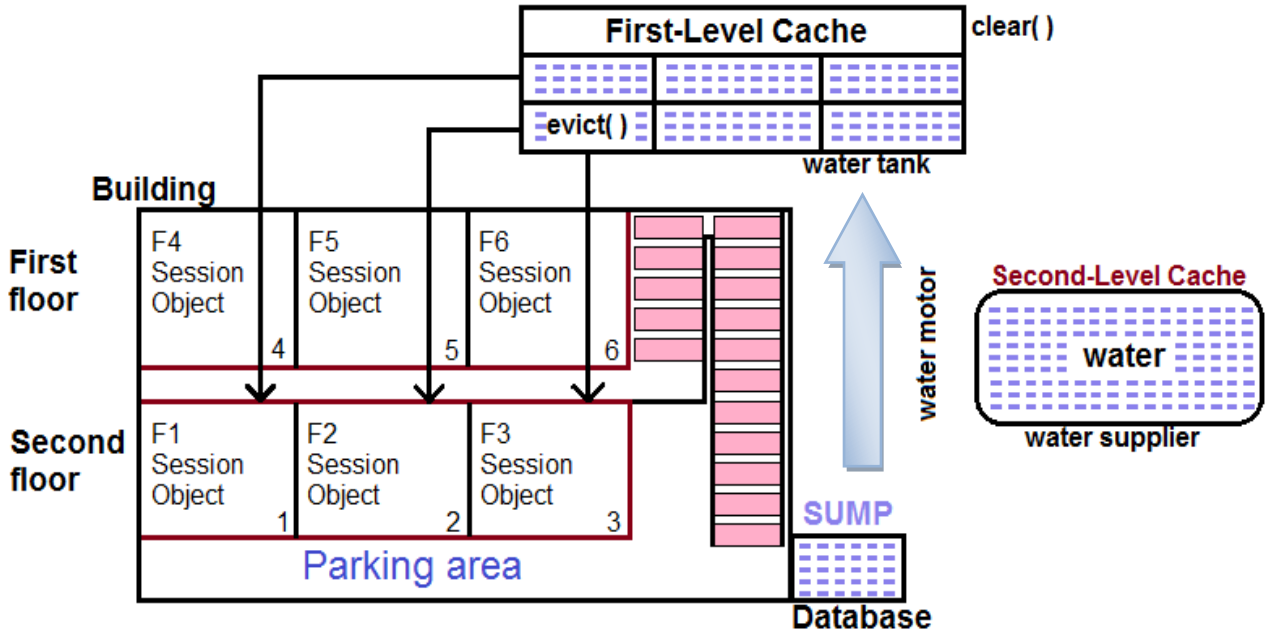
- First-Level Cache is always associated with every Session Object which is enabled by default.
- First-Level Cache is the first cache that the hibernate consults before loading the Object from the Database.
- Once the Session is closed, all the data present in First-Level Cache are cleared.
- There are 2 different methods associated with First-Level Cache namely;
  1. evict( )
  2. clear( )

1. **evict( ):** it is used to remove a particular object from the cache associated with Session
2. **clear( ):** it is used to remove all the Objects from the cache associated with Session

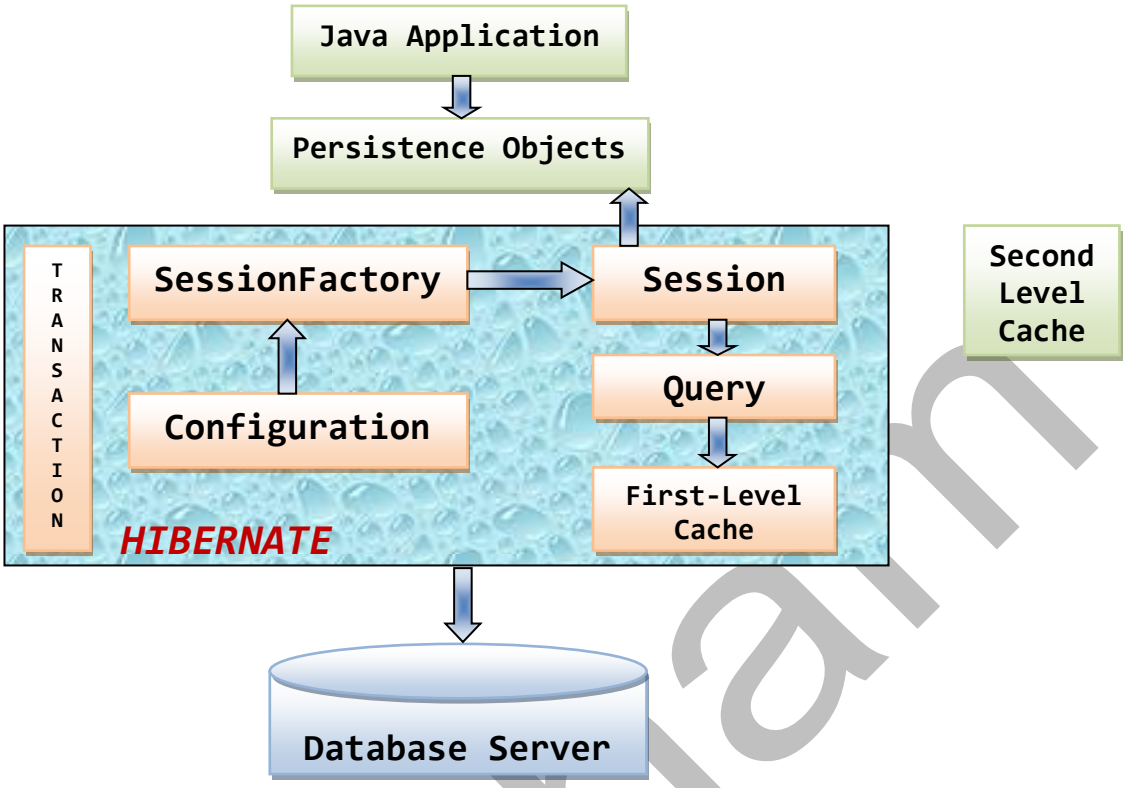
ii. Second-Level Cache

- Second-Level Cache is always associated with SessionFactory Object which is not enabled by default, since it is an Optional cache.
- Since it is an optional cache, the developer has to set up the configuration
- There are different vendors who provides the implementation for second-level cache namely;
  a. EH cache
  b. OS cache
  c. JBOSS cache
  d. Swarm cache etc...

Example:



❖ Hibernate Architecture:



i. **Configuration** is a *class* present in org.hibernate.cfg package which is used to load and validate the configuration file called hibernate.cfg.xml by calling a zero-parameterised **configure( )** method.

```
Configuration conf=new Configuration();
conf.configure();
```

ii. Whenever the configuration file has other name, then we use a parameterised configure( ).

```
conf.configure("Filename.xml");
```

iii. **SessionFactory** is an *interface* present in org.hibernate package for which an implementation object has to be created by using a factory / helper method called **buildSessionFactory( )** along with the reference of Configuration class.

iv. **SessionFactory** is used to establish a connection with the Database server.

v. For each Database server, there is only one **SessionFactory** present.

vi. **SessionFactory** always holds Second-Level Cache.

> Second-Level cache:- It always holds the constant data present in second-level configuration file (data present in SessionFactory Object)

vii. **Session** is an *interface* which is present in org.hibernate package for which an implementation object has to be created by calling a factory / helper method called **openSession( )** along with the reference of **SessionFactory** *interface*.

```
SessionFactory sef=confg.buildSessionFactory();
```

viii. **Session** *object* is light-weight object where 'n' number of Session object can be created for an application along with the same SessionFactory reference.

light-weight object ⇨ number of data present in it is minimum

ix. **Session** is always *Single-Threaded*.

x. **Session** object is responsible to carry the data from the Java Application to the Database server.

xi. **Session** object always holds First-Level cache.

(**)Using **Session** object we perform **CRUD** operations

xii. **Transaction** is an interface present in org.hibernate package for which an implementation object has to created by calling a factory / helper method called beginTransaction( ) along with the reference of **Session** *interface*.

xiii. **Transaction** is used to achieve ***ACID properties*** / ***ACID rules***.

A	Atomicity
C	Consistency
I	Isolation
D	Durability

xiv. **commit( )** is used to save and reflect the changes **tran.commit( );**

✸ ***Instance of Java Bean Class / Entity Class / POJO Class may always exist in anyone of the 3 states;***

1. Transient
2. Persistent
3. Detached

1. **Transient:** Never persistent (not yet saved into Database), not associated with any session.
   Ex: `Student stu=new Student( );`

2. **Persistent:** Associated only with a Unique Session. (to save the object state into the Database)
   Ex: `ses.save(stu);`

3. **Detached:** Previously persistent, not associated with any Session.
   Ex: `ses.close();`

❖ Association / Hibernate Relationship:

> "It represents the relationship between the objects of Java Bean Class / Entity Class / POJO Class"
>
> Or
>
> "It represents the relationship between 2 different tables"

✸ **Need for association:-** Association is needed **to store multiple-entities data** into a ***Single Database Table***

✸ **Problems:-**

1. ***Data Redundancy / Duplication of data***
2. ***Data Maintenance problem***

✸ **Types of Association:-** There are 4 different types of association present namely;

i. One To One
ii. One To Many
iii. Many To One
iv. Many To Many

Example:- `User_Contactnumber(Table)`

PK User_Id	User_name	Address	ContactNumber	Purpose	Provider
420	Uday Kiran	Vijayanagar	99999 99999	office	JIO
420	Uday Kiran	Vijayanagar	99999 99998	home	AIRTEL
420	Uday Kiran	Vijayanagar	99999 99996	Confidential	V!

Problem: - ***Data is Duplicated / Data Redundancy***

Solution:- Create a separate table for User_Details and Create a separate table for Contact_Details

User_Details(Table)

PK User_Id	User_name	Address
420	Uday Kiran	Vijayanagar

Contact_Details(Table)

ContactNumber	Purpose	Provider
99999 99999	office	JIO
99999 99998	home	AIRTEL
99999 99996	Confidential	V!

Problem: - ***Navigation is not possible*** (There is no relationship between these two tables)

Solution: - Build a relationship between the two tables by using a Foreign Key column

User_Details(ParentTable)

PK User_Id	User_name	Address
420	Uday Kiran	Vijayanagar

Contact_Details(ChildTable)

ContactNumber	Purpose	Provider	FK Unid
99999 99999	office	JIO	420
99999 99998	home	AIRTEL	420
99999 99996	Confidential	V!	420

User_Id

✸ **Advantages:-**

1. Data Redundancy problem is solved.
2. Navigation is possible with the help of Foreign Key.
3. Data Maintenance is easy.

Why no Foreign Key in Parent table?
Duplication of data will arise

✸ **Cascade:-**

- Cascade decides the same operation done on the parent object is done with associated child object at the same time
- In all standard association cascade.all( ); is involve

How to differentiate between parent class (table) / child class (table)

- Based on relationship annotation declaration
- for ex:-One Person (child class/table) has One Passport(parent class/table)
  One Person (child class/table) has Many Cars (parent class/table)
  Many Cars (child class/table) belong to One Person (parent class/table)
  Many Students (child class/table) attends Many Courses (parent class/table)

❖ OneToOne relationship:- [One Person has One Passport]

hibernate.cfg.xml

```
<!DOCTYPE hibernate-configuration PUBLIC
      "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
      "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
      <session-factory>
       <!-- database information -->
       <property name="hibernate.connection.driver_class">
            com.mysql.jdbc.Driver</property>
       <property name="hibernate.connection.url">
            jdbc:mysql://localhost:3306/onetoone</property>
       <property name="hibernate.connection.user">
            root</property>
       <property name="hibernate.connection.password">
            root</property>
       <property name="dialect">
            org.hibernate.dialect.MySQL5Dialect</property>
       <property name="show_sql">true</property>
       <property name="hbm2ddl.auto">update</property>
       <mapping class="org.kuk.onetooneApp.Passport"/>
       <mapping class="org.kuk.onetooneApp.Person"/>
    </session-factory>
</hibernate-configuration>
```

Passport.java(Parent class)

```
package org.kuk.onetooneApp;
import javax.persistence.*;
@Entity
@Table
public class Passport
```

```
{
	@Id
	@GeneratedValue
	private int pid;
	private String passportName;
	public int getPid() {
		return pid;
	}
	public void setPid(int pid) {
		this.pid = pid;
	}
	public String getPassportName() {
		return passportName;
	}
	public void setPassportName(String passportName) {
		this.passportName = passportName;
	}
}
```

Person.java(Child class)

```
package org.kuk.onetooneApp;
import javax.persistence.*;
@Entity
@Table
public class Person
{
	@Id
	@GeneratedValue
	private int id;
	private String personname;
	@OneToOne(cascade=CascadeType.ALL)
	private Passport passport;
	public int getId() {
		return id;
	}
	public void setId(int id) {
		this.id = id;
	}
	public String getPersonname() {
		return personname;
	}
	public void setPersonname(String personname) {
		this.personname = personname;
	}
	public Passport getPassport() {
		return passport;
	}
	public void setPassport(Passport passport) {
		this.passport = passport;
	}
}
```

TestDao.java

```
package org.kuk.onetooneApp;
import org.hibernate.*;
import org.hibernate.cfg.Configuration;
public class TestDAO
{
	public static void main(String[] args)
	{
		Configuration conf=new Configuration();
		conf.configure();
		SessionFactory sef=conf.buildSessionFactory();
		Session ses=sef.openSession();
		Transaction trans=ses.beginTransaction();
		Passport passport=new Passport();
		passport.setPassportName("Uday Kiran Kowthalam");
		Person person=new Person();
		person.setPassport(passport);
		person.setPersonname("Uday");
		ses.save(person);
		trans.commit();
		ses.close();
	}
}
```

❖ OneToMany relationship:- [One Person can have Many Cars]

hibernate.cfg.xml

```
<!DOCTYPE hibernate-configuration PUBLIC
	"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
	"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
	<session-factory>
	<!-- database information -->
	<property name="hibernate.connection.driver_class">
		com.mysql.jdbc.Driver</property>
	<property name="hibernate.connection.url">
		jdbc:mysql://localhost:3306/onetomany</property>
	<property name="hibernate.connection.user">
		root</property>
	<property name="hibernate.connection.password">
		root</property>
	<property name="dialect">
		org.hibernate.dialect.MySQL5Dialect</property>
	<property name="show_sql">true</property>
	<property name="hbm2ddl.auto">update</property>
	<mapping class="org.kuk.onetomanyApp.Car"/>
	<mapping class="org.kuk.onetomanyApp.Person"/>
	</session-factory>
</hibernate-configuration>
```

Car.java(Parent class)

```
package org.kuk.onetomanyApp;
import javax.persistence.*;
@Entity
@Table
public class Car {
	@Id
	@GeneratedValue
	private int car_id;
	private String car_name;
	private String car_model;
	public int getCar_id() {
		return car_id;
	}
	public void setCar_id(int car_id) {
		this.car_id = car_id;
	}
	public String getCar_name() {
		return car_name;
	}
	public void setCar_name(String car_name) {
		this.car_name = car_name;
	}
	public String getCar_model() {
		return car_model;
	}
	public void setCar_model(String car_model) {
		this.car_model = car_model;
	}
}
```

Person.java(Child class)

```
package org.kuk.onetomanyApp;
import java.util.List;
import javax.persistence.*;
@Entity
@Table
public class Person {
	@Id
	@GeneratedValue
	private int p_id;
	private String p_name;
	@OneToMany(cascade=CascadeType.ALL)
	private List<Car> cars;
	public int getP_id() {
		return p_id;
	}
	public void setP_id(int p_id) {
		this.p_id = p_id;
	}
```

```
	public String getP_name() {
		return p_name;
	}
	public void setP_name(String p_name) {
		this.p_name = p_name;
	}
	public List<Car> getCars() {
		return cars;
	}
	public void setCars(List<Car> cars) {
		this.cars = cars;
	}
}
```

TestDao.java

```
package org.kuk.onetomanyApp;
import java.util.*;
import org.hibernate.*;
import org.hibernate.cfg.Configuration;
public class TestDao {
	public static void main(String[] args) {
		Configuration conf=new Configuration();
		conf.configure();
		SessionFactory sef=conf.buildSessionFactory();
		Session ses=sef.openSession();
		Transaction tran=ses.beginTransaction();
		Car c1=new Car();
		c1.setCar_name("R8");
		c1.setCar_model("AUDI");

		Car c2=new Car();
		c2.setCar_name("phantom");
		c2.setCar_model("RollsRoyce");

		Car c3=new Car();
		c3.setCar_name("galardo");
		c3.setCar_model("Lambo");

		List<Car> carlist=new ArrayList<Car>();
		carlist.add(c1);
		carlist.add(c2);
		carlist.add(c3);

		Person person=new Person();
		person.setP_name("UdayKiranK");
		person.setCars(carlist);

		ses.save(person);
		tran.commit();
		ses.close();
	}
}
```

❖ ManyToOne relationship:- [Many Cars belong to One Person]

hibernate.cfg.xml

```
<!DOCTYPE hibernate-configuration PUBLIC
      "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
      "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
      <session-factory>
       <!-- database information -->
       <property name="hibernate.connection.driver_class">
            com.mysql.jdbc.Driver</property>
       <property name="hibernate.connection.url">
            jdbc:mysql://localhost:3306/manytoone</property>
       <property name="hibernate.connection.user">
            root</property>
       <property name="hibernate.connection.password">
            root</property>
       <property name="dialect">
            org.hibernate.dialect.MySQL5Dialect</property>
       <property name="show_sql">true</property>
       <property name="hbm2ddl.auto">update</property>
       <mapping class="org.kuk.manytooneApp.Person"/>
       <mapping class="org.kuk.manytooneApp.Car"/>
      </session-factory>
</hibernate-configuration>
```

Person.java(Parent Class)

```
package org.kuk.manytooneApp;
import javax.persistence.*;
@Entity
@Table(name="Person_details")
public class Person {
      @Id
      @GeneratedValue
      private int person_id;
      private String personName;
      public int getPerson_id() {
            return person_id;
      }
      public void setPerson_id(int person_id) {
            this.person_id = person_id;
      }
      public String getPersonName() {
            return personName;
      }
      public void setPersonName(String personName) {
            this.personName = personName;
      }
}
```

Cars.java(Child class)

```
package org.kuk.manytooneApp;
import javax.persistence.*;
@Entity
@Table(name="Car_details")
public class Car {
	@Id
	@GeneratedValue
	private int car_id;
	private String carName;
	private String carModel;
	@ManyToOne(cascade=CascadeType.ALL)
	private Person person;
	public Person getPerson() {
		return person;
	}
	public void setPerson(Person person) {
		this.person = person;
	}
	public int getCar_id() {
		return car_id;
	}
	public void setCar_id(int car_id) {
		this.car_id = car_id;
	}
	public String getCarName() {
		return carName;
	}
	public void setCarName(String carName) {
		this.carName = carName;
	}
	public String getCarModel() {
		return carModel;
	}
	public void setCarModel(String carModel) {
		this.carModel = carModel;
	}
}
```

TestDao.java

```
package org.kuk.manytooneApp;
import org.hibernate.*;
import org.hibernate.cfg.Configuration;
public class TestDao {
	public static void main(String[] args) {
		Configuration conf=new Configuration();
		conf.configure();
		SessionFactory sef=conf.buildSessionFactory();
```

```
            Session ses=sef.openSession();
            Transaction trans=ses.beginTransaction();
            Person person=new Person();
            person.setPersonName("Uday Kiran K");

            Car c1=new Car();
            c1.setCarName("Phantom");
            c1.setCarModel("RollsRoyce");
            c1.setPerson(person);

            Car c2=new Car();
            c2.setCarName("Q7");
            c2.setCarModel("AUDI");
            c2.setPerson(person);

            Car c3=new Car();
            c3.setCarName("Chiron");
            c3.setCarModel("Bugati");
            c3.setPerson(person);

            ses.save(c1);
            ses.save(c2);
            ses.save(c3);
            trans.commit();
            ses.close();
      }
}
```

❖ ManyToMany relationship:- [Many Students attend Many Courses]

hibernate.cfg.xml

```
<!DOCTYPE hibernate-configuration PUBLIC
      "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
      "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
      <session-factory>
       <!-- database information -->
       <property name="hibernate.connection.driver_class">
            com.mysql.jdbc.Driver</property>
       <property name="hibernate.connection.url">
            jdbc:mysql://localhost:3306/manytomany</property>
       <property name="hibernate.connection.user">
            root</property>
       <property name="hibernate.connection.password">
            root</property>
       <property name="dialect">
            org.hibernate.dialect.MySQL5Dialect</property>
       <property name="show_sql">true</property>
       <property name="hbm2ddl.auto">update</property>
       <mapping class="org.kuk.manytomanyApp.Course"/>
       <mapping class="org.kuk.manytomanyApp.Student"/>
```

```
	</session-factory>
</hibernate-configuration>
```

Course.java(Parent class)

```
package org.kuk.manytomanyApp;
import javax.persistence.*;
@Entity
@Table(name="Course_details")
public class Course {
	@Id
	@GeneratedValue
	private int course_id;
	private String courseName;
	public int getCourse_id() {
		return course_id;
	}
	public void setCourse_id(int course_id) {
		this.course_id = course_id;
	}
	public String getCourseName() {
		return courseName;
	}
	public void setCourseName(String courseName) {
		this.courseName = courseName;
	}
}
```

Student.java(Child class)

```
package org.kuk.manytomanyApp;
import java.util.List;
import javax.persistence.*;
@Entity
@Table(name="Student_details")
public class Student {
	@Id
	@GeneratedValue
	private int student_id;
	private String student_name;
	@ManyToMany(cascade=CascadeType.ALL)
	private List<Course> listofcourses;
	public int getStudent_id() {
		return student_id;
	}
	public void setStudent_id(int student_id) {
		this.student_id = student_id;
	}
	public String getStudent_name() {
		return student_name;
	}
	}
```

```
	public void setStudent_name(String student_name) {
		this.student_name = student_name;
	}
	public List<Course> getListofcourses() {
		return listofcourses;
	}
	public void setListofcourses(List<Course> listofcourses) {
		this.listofcourses = listofcourses;
	}
}
```

TestDao.java

```
package org.kuk.manytomanyApp;
import java.util.*;
import org.hibernate.*;
import org.hibernate.cfg.Configuration;
public class TestDao {
	public static void main(String[] args) {
		Configuration conf=new Configuration();
		conf.configure();
		SessionFactory sef=conf.buildSessionFactory();
		Session ses=sef.openSession();
		Transaction tran=ses.beginTransaction();
		Course c1=new Course();
		c1.setCourseName("Java");

		Course c2=new Course();
		c2.setCourseName("J2EE");

		Course c3=new Course();
		c3.setCourseName("Framework");

		Student s1=new Student();
		s1.setStudent_name("Uday");

		Student s2=new Student();
		s2.setStudent_name("Kiran");

		List<Course> course=new ArrayList<Course>();
		course.add(c1);
		course.add(c2);
		course.add(c3);

		List<Student> student=new ArrayList<Student>();
		student.add(s1);
		student.add(s2);

		s1.setListofcourses(course);
		s2.setListofcourses(course);

		ses.save(s1);
```

```
            ses.save(s2);
            tran.commit();
            ses.close();
        }
}
```

In case of OneToMany & ManyToMany, the third table will be generated by merging Parent Table and Child Table which will have 2 columns;
1- Foreign Key column 2-Primary Key column

❖ Hibernate interview question:-

1) What are the Drawbacks of JDBC?

- In all the cases, all the steps will remain the same except the 4th step; this condition is called **Duplication of code** (or) **Boiler Plate Code.**
- To avoid this **"Duplication of code** (or) **Boiler Plate Code"**, we use **SingleTon Class.**
- Where **SingleTon** class always returns 2 methods

c) for Connection ⇒ Trying to establish a connection with the DB server

d) closing the Connection ⇒ To close the Connection [since it is a costly resource]

✸ **JDBC is always dependent on Database.**

[Since the SQL queries varies from one Database Server to another.]

- Hibernate is independent of Database, since hibernate automatically generates SQL queries as per the Database

✸ **JDBC does not support auto-generation of Tables & Primary Key.**

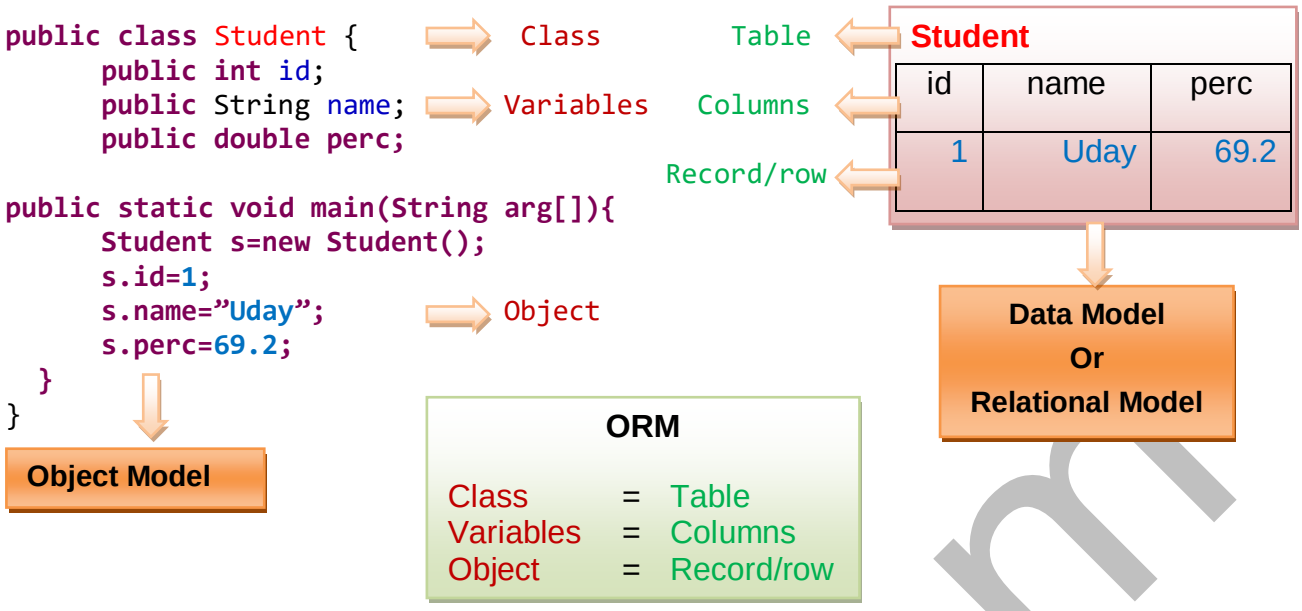
- In case of hibernate, hibernate will automatically generate the tables on its own along with assigning primary key.

✸ **JDBC does not support "cache mechanism"**

"**Cache memory**":- It is a temporary (or) buffer area, which stores constant data / repeated data from the Database.

(**) cache mechanism avoids traffic between Java Application and Database Server, hence performance of an application increases.

2. What is ORM and mention the tools and specifications of ORM?



Object Model

Why it is called as Object Model?

Until we create an Object we cannot access one single functionality

Data Model (or) Relational Model

Why it is called as Data Model?

Apart from data we don't find any other components

"Conversion of Java Object into Relational Model is known as Object Relational Mapping".

Or

"The Object which created in case of Java must mandatorily have a relationship with data's in the Database server is known as **Object Relational Mapping** [**ORM**]".

- There are different types of mechanism / tools present in ORM namely;
  - iv. Hibernate
  - v. TopLink
  - vi. Ibatis etc...

❖ Specifications of ORM:-

There are 3 different specifications wrt ORM;

iv. Every Java Bean Class / Entity Class / POJO class represents a Table.

v. Every Variable of Java Bean Class represents a Column.

vi. Every Java Bean Class Object represents a Record/row.

ORM

Class = Table
Variables = Columns
Object = Record/row

3. Define Framework and explain the categories of Framework?

- Framework is a set of API's (or) semi-software (pre-written code) which is used to develop an application in a loosely coupled manner.
  Ex.:- Ready-made template
- It is used to develop an application in a simplified manner (or) using which we can develop an application in a simplified manner.
- There are 2 different categories present in framework;
  i. Invasive framework
  ii. Non-invasive framework

i. Invasive Framework:-

"***Framework which allows to*** extends one of their classes / implements one of their interfaces".

Ex.:- Struts, EJB framework

ii. Non-invasive Framework:-

"***Framework which does not allow to*** extend one of their classes / implements one of their interfaces".

Ex.:- Hibernate and Springs

4. Difference between API and Framework?

(***)Difference between API and Framework

:API:	:FRAMEWORK:
Is an interface between two different application Ex.: Java-Collection	Is used to design / develop application such as MVC web application Ex: Java-Array

5. Define Hibernate?

"Hibernate is an open-source, non-invasive framework, ORM tool which is used to convert Java Model [classes & variables] into Relational Model [Tables & Columns]"

6**. What are the advantages of Hibernate?**

a. Hibernate is independent of Database.
   [Since SQL queries are automatically generated by hibernate as per the Database]
b. Hibernate supports auto-generation of Tables and Primary key.
c. Hibernate supports cache mechanism (avoids traffic between java application and Database server).

d. Hibernate supports Connection Pooling (resources which stores Database connection)
   [i.e., from one single java application we can hit multiple Database servers at a time]
e. Hibernate supports HQL [Hibernate Query Language]
f. (***) Hibernate supports dialect

7. Define the properties of Dialect?

Dialect: - **i)** It is responsible for generating SQL queries based on each Database

ii) To achieve Object Relational Mapping.

8. What are the pre-requisites for Hibernate Java Application?

a. Open Java Perspective and select *Navigator* mode
b. Right click within Navigator mode and create a *new* Java Project & name it
c. Right click on Project and create a new folder called lib and add all the Jar files into the lib folder and build a java path to import the properties from the jar file.
d. Add the configuration file [hibernate.cfg.xml] into the source folder
e. Select src folder and create a new package structure
f. Select application name and create a class [java bean class / entity class / POJO class]

9. Explain the different JPA Annotations?

@Entity	specifies it is an Entity class which is used to write Hibernate logic
@Table	specifies Table name in the Database which maps Entity [Table name will be associated with Entity class]
@Id	specifies it is Primary Key where the data type must be int or long
@GeneratedValue	specifies auto-generation of Primary Key [because of this hibernate supports auto-generation of PK]

10. What is the use of DAO class?

DAO class **Data Access Object**

Why we use DAO class?

- Since DAO class is the only area where we can access the properties or functionalities of Java Bean Class / Entity Class / POJO Class
- Only class / area to execute java application which has main method in it. [JVM is responsible for execution]

11. Difference between get( ) and load( ) ?

get( )	load( )
it directly hits the DB server and returns the real object/actual object	It always returns a Proxy object without hitting the DB server
If the ID is not present in the table, then get( ) throws NullPointerException (or) NULL	If the ID is not present in the table, then load( ) throws ObjectNotFoundException

12. Define Proxy Object?

Proxy object is an Object with a given ***Id*** but the properties are not initialized yet

13. What are the steps to be followed in DAO class?

a. Create an object of **Java Bean Class** / **Entity class** / **POJO class**
b. Set the value of the members of Java Bean Class or Entity class
c. Create an object of Configuration class present in org.hibernate.cfg package
***Syntax*:**

```
Configuration conf =new Configuration();
```

d. Call a zero-parameterised **configure()** method which is declared in Configuration Class.
***Syntax*:**

```
conf =configure();
```

> Why to call a Zero-parameterised **configure()** ?
> **configure()** is responsible for loading and validating the configuration file called hibernate.cfg.xml

e. Create an implementation object of **SessionFactory** *interface* present in org.hibernate package by using a factory / helper method called *buildSessionFactory( )* which is declared inside Configuration class
***Syntax*:**

```
SessionFactory sef=conf.buildSessionFactory();
```

f. Create an implementation object of **Session** *interface* present in org.hibernate package by using a factory / helper method called *openSession( )* which is declared inside **SessionFactory** *interface*
***Syntax*:**

```
Session ses= sef.openSession();
```

g. Create an implementation object of **Transaction** *interface* present in org.hibernate package by using a factory / helper method called *beginTransaction( )* which is declared inside **Session** *interface*
***Syntax*:**

```
Transaction tran= ses.beginTransaction();
```

h. Perform **CRUD** operation by using the reference of **Session**

***Syntax*:**

```
ses.save(Object);
```

`save(Object)` is declared in **Session** *interface* responsible to save / insert data's into the Database (only objects of java bean class)

i. Commit the **Transaction** in order **to save** the data into the Database server

Syntax:

```
tran.commit()
```

j. Close the Session

Since the Session is considered to be a costly resource, so we need to close the costly resource

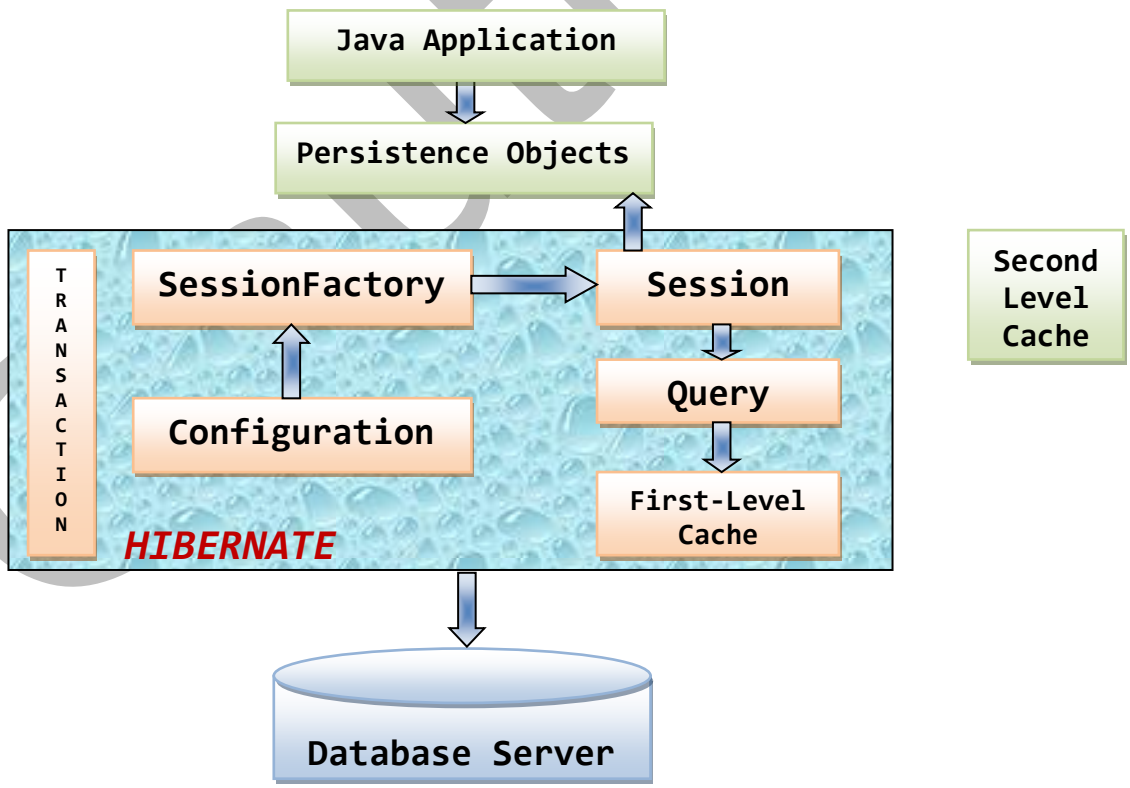
Syntax:

```
ses.close()
```

14. What is the role of configure( ) ?

`configure()` is responsible for loading and validating the configuration file called hibernate.cfg.xml

15. Explain hibernate architecture in brief?



i. **Configuration** is a *class* present in org.hibernate.cfg package which is used to load and validate the configuration file called hibernate.cfg.xml by calling a zero-parameterised **configure( )** method.

```
Configuration conf=new Configuration();
conf.configure();
```

ii. Whenever the configuration file has other name, then we use a parameterised configure( ).

```
conf.configure("Filename.xml");
```

iii. **SessionFactory** is an *interface* present in org.hibernate package for which an implementation object has to be created by using a factory / helper method called **buildSessionFactory( )** along with the reference of Configuration class.

iv. **SessionFactory** is used to establish a connection with the Database server.

v. For each Database server, there is only one **SessionFactory** present.

vi. **SessionFactory** always holds Second-Level Cache.

> Second-Level cache:- It always holds the constant data present in second-level configuration file (data present in SessionFactory Object)

vii. **Session** is an *interface* which is present in org.hibernate package for which an implementation object has to be created by calling a factory / helper method called **openSession( )** along with the reference of **SessionFactory** *interface*.

```
SessionFactory sef=confg.buildSessionFactory();
```

viii. **Session** *object* is light-weight object where 'n' number of Session object can be created for an application along with the same SessionFactory reference.

light-weight object ⇨ number of data present in it is minimum

ix. **Session** is always *Single-Threaded*.

x. **Session** object is responsible to carry the data from the Java Application to the Database server.

xi. **Session** object always holds First-Level cache.

> (**)Using **Session** object we perform **CRUD** operations

xii. **Transaction** is an interface present in org.hibernate package for which an implementation object has to created by calling a factory / helper method called beginTransaction( ) along with the reference of **Session** *interface*.

xiii. **Transaction** is used to achieve ***ACID properties*** / ***ACID rules***.

A	Atomicity
C	Consistency
I	Isolation
D	Durability

xiv. **commit( )** is used to save and reflect the changes **tran.commit( )**

16. Explain cache mechanism in brief?

Cache memory: - it is temporary / buffer area which stores constant data or repeated data from the Database

Storage of cache memory: cache memory is generally stored in **RAM**

✸ **Advantage of cache mechanism:-**

- Cache mechanism avoids the traffic between Java application and Database server.
  Hence, the performance of an application increases.

There are 2 different types of cache present namely;

i. First-Level Cache
ii. Second-Level Cache

i. First-Level Cache

- First-Level Cache is always associated with every Session Object which is enabled by default.
- First-Level Cache is the first cache that the hibernate consults before loading the Object from the Database.
- Once the Session is closed, all the data present in First-Level Cache are cleared.
- There are 2 different methods associated with First-Level Cache namely;
  1. evict( )
  2. clear( )

1. **evict( ):** it is used to remove a particular object from the cache associated with Session
2. **clear( ):** it is used to remove all the Objects from the cache associated with Session

ii. Second-Level Cache

- Second-Level Cache is always associated with SessionFactory Object which is not enabled by default, since it is an Optional cache.
- Since it is an optional cache, the developer has to set up the configuration
- There are different vendors who provides the implementation for second-level cache namely;
  e. EH cache
  f. OS cache
  g. JBOSS cache
  h. Swarm cache etc...

17. Difference between evict( ) and clear( ) ?

evict( )	**clear( )**
It is used to remove a particular object from the cache associated with Session	It is used to remove all the objects from the cache associated with Session

18. Define the Generator and explain the types of Generator?

- "**Generator** is the one which is used to generate a primary key value based on Generation strategy / Generation Algorithm".
- It is a responsible for auto-generation of Primary Key.
- There are 2 different categories of generator present;
  i. JPA generator
  ii. Hibernate generator

i. JPA Generator:- JPA generator supports 4 different types of Primary Key Generation Strategy which are as follows;

a. GenerationType.**AUTO**
b. GenerationType.**IDENTITY**
c. GenerationType.**SEQUENCE**
d. GenerationType.**TABLE**

a. GenerationType.**AUTO:**

- It is a ***default*** GenerationType which selects the generation strategy based on Database specific **dialect**
  [Based on the respective Database specifications, according to that primary key will be generated automatically]

Why default GenerationType**:-** For 2 important reasons;

1. Since it is supported by all the Database servers. Hence, the name ***default*** GenerationType
2. Whenever we **don't mention** the GenerationType, by ***default*** the GenerationType will be considered as GenerationType**.AUTO**

Syntax:

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

- Whenever we delete the data from actual table the record will not be deleted from comparison table
- Whenever we add new record on actual table the record will be added into comparison table

Comparison Table	Actual Table
1 ✓	1 ✓
2 ✓	2 ✓
3 ✓	3 ✗
4 ✓	4 ✓
	USER

b. GenerationType.**IDENTITY**:

- This GenerationType relies on an auto-incremented Database column

Syntax:

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

- It will check id present in Database and then will automatically increment the value based on that id in Database

c. GenerationType.**SEQUENCE**:

- This GenerationType is responsible for generating a primary key based on **sequence algorithm**
- This GenerationType is supported by only Oracle, IBM DB2, Postgres Database server

Syntax:

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

d. GenerationType.**TABLE**:

- This GenerationType is responsible for generating a primary key based on **Table algorithm**

Syntax:

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

```
1st value of id to be inserted id=1 for the 1st time
2nd time   : id= 32768
3rd time   : id= 65536
4th time   : id=98304
5th time   : id=131072
```

❖ Hibernate Generator:

- It supports many types of primary key Generation strategy which are as follows;
  a. Increment
  b. Foreign
  c. Identity
  d. Sequence
  e. Hilo
  f. Seqhilo
  g. Uuid
  h. Guid
  i. Assigned etc...

a. ***Increment***:-

- It generates the identifiers of type int, long or short that is unique where no other process is inserting the data into the same table

- Increment always auto-increments the value of Primary Key based on the maximum value of primary key present in the table
  [Maximum value of **Primary Key + 1**]

Syntax:

```
@Id
@GeneratedValue(generator="mygen")
@GenericGenerator(name="mygen",strategy= "increment")
```

b. ***Foreign***:-

It uses the identifiers of another associated object which is generally used in conjunction with **<OneToOne>** Primary Key Association

19. Define Association or Hibernate relationship?

"It represents the relationship between the objects of Java Bean Class / Entity Class / POJO Class"

Or

"It represents the relationship between 2 different tables"

✸ **Need for association:-** Association is needed **to store multiple-entities data** into a ***Single Database Table***

✸ **Problems:-**

1. ***Data Redundancy / Duplication of data***
2. ***Data Maintenance problem***

✸ **Types of Association:-** There are 4 different types of association present namely;

i. One To One
ii. One To Many
iii. Many To One
iv. Many To Many

20. What are the advantages of Association?

a. Data Redundancy problem is solved.
b. Navigation is possible with the help of Foreign Key.
c. Data Maintenance is easy