UNIT: 4 MVC ARCHITECTURE,EJB, HIBERNATE

CS-25 Advanced Java Programming (J2EE)

- **RMI AND RMI ARCHITECTURE**
- **INTRODUCTION TO MVC**
- **IMPLEMENTATION OF MVC ARCHITECTURE**
- **INTRODUCTION & BENEFITS OF EJB**
- **RESTRICTION ON EJB**
- **TYPES OF EJB**
- **SESSION BEANS**
- **ENTITY BEANS**
- **MESSAGE-DRIVEN BEANS**
- **TIMER SERVICE**
- **INTRODUCTION TO HIBERNATE**
- **NEED FOR HIBERNATE**
- **FEATURES OF HIBERNATE**
- **DISADVANTAGES OF HIBERNATE**
- **EXPLORING HIBERNATE ARCHITECTURE**
- **DOWNLOADING AND CONFIGURING AND NECESSARY FILES TO HIBERNATE IN ECLIPSE**
- **JARS FILES OF HIBERNATE.**
- **HIBERNATE CONFIGURATION FILE**
- **HIBERNATE MAPPING FILE**
- **BASIC EXAMPLE OF HIBERNATE**
- **ANNOTATION**
- **HIBERNATE INHERITANCE**
- **INHERITANCE ANNOTATIONS**
- **HIBERNATE SESSIONS**

: ASSIGNMENT 4:

- JNDI stands for ?
- The __ annotation is used to specify the details of the column.
- ORM stands for?
- Explain features of hibernate.
- Explain hibernate session.
- Explain in brief MVC architecture.
- MVC stands for ?
- A _bean is an enterprise bean that allows j2ee applications to process messages asynchronously.
- __ powerful, high-performance object relational persistence and query service for any java application.
- JTA stands for ?
- Explain message driven beans.
- Explain timer service.
- What is hibernate ? Explain hibernate architecture.
- Explain @id and @entity hibernate notation
- Stateful session bean vs stateless session bean.
- What is hibernate explain with advantages and disadvantages.
- Explain disadvantages of EJB.
- Explain saveorupdate() and getidentifire methods.

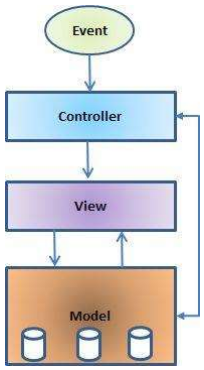
Introduction to MVC

Model View Controller or MVC as it is popularly called, is a software design pattern for developing web applications. A Model View Controller pattern is made up of the following three parts:

Model - The lowest level of the pattern which is responsible for maintaining data.

View - This is responsible for displaying all or a portion of the data to the user.

Controller - Software Code that controls the interactions between the Model and View.



MVC is popular as it isolates the application logic from the user interface layer and supports separation of concerns. Here the Controller receives all requests for the application and then works with the Model to prepare any data needed by the View. The View then uses the data prepared by the Controller to generate a final presentable response. The MVC abstraction can be graphically represented as follows.

MVC architecture

The Model-View-Controller (MVC) architecture is a widely used architectural approach for interactive applications that distributes functionality among application objects so as to minimize the degree of coupling between the objects. To achieve this, it divides applications into three layers: model, view, and controller. Each layer handles specific tasks and has responsibilities to the other layers:

Model

The model is responsible for managing the data of the application. It responds to the request from the view and it also responds to instructions from the controller to update itself. The model represents business data, along with business logic or operations that govern access and modification of this business data. The model notifies views when it changes and lets the view query the model about its state. It also lets the controller access application functionality encapsulated by the model. In the Duke's Bookstore application, the shopping cart and database access object contain the business logic for the application.

View

A presentation of data in a particular format, triggered by a controller's decision to present the data. They are script based templating systems like JSP, ASP, PHP and very easy to integrate with AJAX technology. The view renders the contents of a model. It gets data from the model and specifies how that data should be presented. It updates data presentation when the model changes. A view also forwards user input to a controller. The Duke's Bookstore JSP pages format the data stored in the session scoped shopping cart and the page-scoped database bean.

Controller

The controller is responsible for responding to user input and performs interactions on the data model objects. The controller receives the input; it validates the input and then performs the business operation that modifies the state of the data model. The controller defines application behaviour. It dispatches user requests and selects views for presentation. It interprets user inputs and maps them into actions to be performed by the model. In a web application, user inputs are HTTP GET and POST requests. A controller selects the next view to display based on the user interactions and the outcome of the model operations. In the Duke's Bookstore application, the Dispatcher servlet is the controller. It examines the request URL, creates and initializes a session scoped JavaBeans component the shopping cart and dispatches requests to view JSP pages.

Introduction of EJB

EJB stands for Enterprise Java Beans. EJB is an essential part of a J2EE platform. J2EE platform have component based architecture to provide multi-tiered, distributed and highly transactional features to enterprise level applications. EJB provides an architecture to develop and deploy component based enterprise applications considering robustness, high scalability and high performance. An EJB application can be deployed on any of the application server compliant with J2EE 1.3 standard specification. We'll be discussing EJB 3.0 in this tutorial. ENTERPRISE beans are the J2EE components that implement Enterprise Java-Beans (EJB) technology. Enterprise beans run in the EJB container, a runtime environment within the Application Server. Although transparent to the application developer, the EJB container provides system-level services such as transactions and security to its enterprise beans. These services enable you to quickly build and deploy enterprise beans, which form the core of transactional J2EE applications.

What Is an Enterprise Bean

Written in the Java programming language, an enterprise bean is a server-side component that encapsulates the business logic of an application. The business logic is the code that full fills the purpose of the application. In an inventory control application, for example, the enterprise beans might implement the business logic in methods called checkInventoryLevel and orderProduct. By invoking these methods, remote clients can access the inventory services provided by the application.

Benefits of EJB

Simplified development of large scale enterprise level application. Application Server/ EJB container provides most of the system level services like transaction handling, logging, load balancing, persistence mechanism, exception handling and so on. Developer has to focus only on business logic of the application. EJB container manages life cycle of ejb instances thus developer needs not to worry about when to create/delete ejb objects. For several reasons, enterprise beans simplify the development of large, distributed applications.

First, because the EJB container provides system-level services to enterprise beans, the bean developer can concentrate on solving business problems. The EJB container and not the bean developer are responsible for system-level services such as transaction management and security authorization.

Second, because the beans and not the clients contain the application's business logic, the client developer can focus on the presentation of the client. The client developer does not have to code the routines that implement business rules or access databases. As a result, the clients are thinner, a benefit that is particularly important for clients that run on small devices.

Third, because enterprise beans are portable components, the application assembler can build new applications from existing beans. These applications can run on any compliant J2EE server provided that they use the standard APIs.

Restriction on Enterprise Beans

You should consider using enterprise beans if your application has any of the following requirements:

• The application must be scalable. To accommodate a growing number of users, you may need to distribute an application's components across multiple machines. Not only can the enterprise beans of an application run on different machines, but also their location will remain transparent to the clients.

• Transactions must ensure data integrity. Enterprise beans support transactions, the mechanisms that manage the concurrent access of shared objects.

• The application will have a variety of clients. With only a few lines of code, remote clients can easily locate enterprise beans. These clients can be thin, various, and numerous.

Types of EJB

Type	Description
Session Bean	Session bean stores data of a particular user for a single session. It can be stateful or stateless. It is less resource intensive as compared to entity beans. Session bean gets destroyed as soon as user session terminates.
Entity Bean	Entity beans represent persistent data storage. User data can be saved to database via entity beans and later on can be retrieved from the database in the entity bean.
Message Driven Bean	Message driven beans are used in context of JMS (Java Messaging Service). Message Driven Beans can consumes JMS messages from external entities and act accordingly.

- **Session** Performs a task for a client; implements a web service.
- **Entity** Represents a business entity object that exists in persistent storage.
- **Message-Driven** Acts as a listener for the Java Message Service API, processing messages asynchronously.

Session Bean

A session bean represents a single client inside the Application Server. To access an application that is deployed on the server, the client invokes the session bean's methods. The session bean performs work for its client, shielding the client from complexity by executing business tasks inside the server. As its name suggests, a session bean is similar to an interactive session. A session bean is not shared; it can have only one client, in the same way that an interactive session can have only one user. Like an interactive session, a session bean is not persistent. (That is, its data is not saved to a database.) When the client terminates, its session bean appears to terminate and is no longer associated with the client.

State Management Modes

There are two types of session beans: stateless and stateful.

Stateless Session Beans

A stateless session bean does not maintain a conversational state for the client. When a client invokes the method of a stateless bean, the bean's instance variables may contain a state, but only for the duration of the invocation. When the method is finished, the state is no longer retained. Except during method invocation, all instances of a stateless bean are equivalent, allowing the EJB container to assign an instance to any client. Because stateless session beans can support multiple clients, they can offer better scalability for applications that require large numbers of clients. Typically, an application requires fewer stateless session beans than stateful session beans to support the same number of clients. At

times, the EJB container may write a stateful session bean to secondary storage. However, stateless session beans are never written to secondary storage. Therefore, stateless beans may offer better performance than stateful beans. A stateless session bean can implement a web service, but other types of enterprise beans cannot.

Stateful Session Beans
The state of an object consists of the values of its instance variables. In a stateful session bean, the instance variables represent the state of a unique client-bean session. Because the client interacts ("talks") with its bean, this state is often called the conversational state. The state is retained for the duration of the client-bean session. If the client removes the bean or terminates, the session ends and the state disappears. This transient nature of the state is not a problem, however, because when the conversation between the client and the bean ends there is no need to retain the state.

When to Use Session Beans
In general, you should use a session bean if the following circumstances hold:

- At any given time, only one client has access to the bean instance.
- The state of the bean is not persistent, existing only for a short period (perhaps a few hours).
- The bean implements a web service.
- Stateful session beans are appropriate if any of the following conditions are true:
    - The bean's state represents the interaction between the bean and a specific Client.
    - The bean needs to hold information about the client across method invocations.
    - The bean mediates between the client and the other components of the application, presenting a simplified view to the client.
- Behind the scenes, the bean manages the work flow of several enterprise beans.
- To improve performance, you might choose a stateless session bean if it has any Of these traits
    - The bean's state has no data for a specific client.
    - In a single method invocation, the bean performs a generic task for all clients. For example, you might use a stateless session bean to send an email that confirms an online order.
    - The bean fetches from a database a set of read-only data that is often used by clients. Such a bean, for example, could retrieve the table rows that represent the products that are on sale this month.

Entity Bean
An entity bean represents a business object in a persistent storage mechanism. Some examples of business objects are customers, orders, and products. In the Application Server, the persistent storage mechanism is a relational database. Typically, each entity bean has an underlying table in a relational database, and each instance of the bean corresponds to a row in that table.

What Makes Entity Beans Different from Session Beans?
Entity beans differ from session beans in several ways. Entity beans are persistent, allow shared access, have primary keys, and can participate in relationships with other entity beans.

Persistence
Because the state of an entity bean is saved in a storage mechanism, it is persistent. Persistence means that the entity bean's state exists beyond the lifetime of the application or the Application Server process. If you've worked with databases, you're familiar with persistent data. The data in a database is persistent because it still exists even after you shut down the database server or the applications it services.

There are two types of persistence for entity beans: bean-managed and container managed.

With bean-managed persistence, the entity bean code that you write contains the calls that access the database. If your bean has container-managed persistence, the EJB container automatically generates the necessary database access calls. The code that you write for the entity bean does not include these calls. For additional information, see the section Container-Managed Persistence.

Shared Access

Entity beans can be shared by multiple clients. Because the clients might want to change the same data, it's important that entity beans work within transactions. Typically, the EJB container provides transaction management. In this case, you specify the transaction attributes in the bean's deployment descriptor. You do not have to code the transaction boundaries in the bean; the container marks the boundaries for you.

Primary Key

Each entity bean has a unique object identifier. A customer entity bean, for example, might be identified by a customer number. The unique identifier, or primary key, enables the client to locate a particular entity bean. For more information, see the section Primary Keys for Bean-Managed Persistence.

Relationships

Like a table in a relational database, an entity bean may be related to other entity beans. For example, in a college enrolment application, StudentBean and CourseBean would be related because students enroll in classes. You implement relationships differently for entity beans with bean-managed persistence than those with container-managed persistence. With bean-managed persistence, the code that you write implements the relationships. But with container- managed persistence, the EJB container takes care of the relationships for you. For this reason, relationships in entity beans with container-managed persistence are often referred to as container-managed relationships.

When to Use Entity Beans

You should probably use an entity bean under the following conditions:

- The bean represents a business entity and not a procedure. For example, CreditCardBean would be an entity bean, but CreditCardVerifierBean would be a session bean.
- The bean's state must be persistent. If the bean instance terminates or if the Application Server is shut down, the bean's state still exists in persistent storage (a database).

Message-Driven Bean

A message-driven bean is an enterprise bean that allows J2EE applications to process messages asynchronously. It normally acts as a JMS message listener, which is similar to an event listener except that it receives JMS messages instead of events. The messages can be sent by any J2EE component an application client, another enterprise bean, or a web component or by a JMS application or system that does not use J2EE technology. Message-driven beans can process either JMS messages or other kinds of messages.

What Makes Message-Driven Beans Different from Session and Entity Beans?

The most visible difference between message-driven beans and session and entity beans is that clients do not access message-driven beans through interfaces. Interfaces are described in the section Defining

Client Access with Interfaces. Unlike a session or entity bean, a message-driven bean has only a bean class.

In several respects, a message-driven bean resembles a stateless session bean.

- A message-driven bean's instances retain no data or conversational state for a specific client.
- All instances of a message-driven bean are equivalent, allowing the EJB container to assign a message to any message-driven bean instance. The container can pool these instances to allow streams of messages to be processed concurrently.
- A single message-driven bean can process messages from multiple clients. The instance variables of the message-driven bean instance can contain some state across the handling of client messages—for example, a JMS API connection, an open database connection, or an object reference to an enterprise bean object.
- Client components do not locate message-driven beans and invoke methods directly on them. Instead, a client accesses a message-driven bean through JMS by sending messages to the message destination for which the message-driven bean class is the MessageListener. You assign a message-driven bean's destination during deployment by using Application Server resources.
- **Message-driven beans have the following characteristics:**
  - They execute upon receipt of a single client message.
  - They are invoked asynchronously.
  - They are relatively short-lived.
  - They do not represent directly shared data in the database, but they can access and update this data.
  - They can be transaction-aware.
  - They are stateless.
- When a message arrives, the container calls the message-driven bean's onMessage method to process the message.
- The onMessage method normally casts the message to one of the five JMS message types and handles it in accordance with the application's business logic. The onMessage method can call helper methods, or it can invoke a session or entity bean to process the information in the message or to store it in a database.
- A message can be delivered to a message-driven bean within a transaction context, so all operations within the onMessage method are part of a single transaction. If message processing is rolled back, the message will be redelivered.

When to Use Message-Driven Beans

Session beans and entity beans allow you to send JMS messages and to receive them synchronously, but not asynchronously. To avoid tying up server resources, you may prefer not to use blocking synchronous receives in a server-side component. To receive messages asynchronously, use a message-driven bean.

Timer Service

Applications that model business work flows often rely on timed notifications. The timer service of the enterprise bean container enables you to schedule timed notifications for all types of enterprise beans except for stateful session beans. You can schedule a timed notification to occur at a specific time, after duration of time, or at timed intervals. For example, you could set timers to go off at 10:30 AM on May 23, in 30 days, or every 12 hours.

When a timer expires (goes off), the container calls the ejbTimeout method of the bean's implementation class. The ejb Timeout method contains the business logic that handles the timed event. Because

ejbTimeout is defined by the javax.ejb.TimedObject interface, the bean class must implement TimedObject.

There are four interfaces in the javax.ejb package that are related to timers:

- TimedObject
- Timer
- TimerHandle
- TimerService

Creating Timers

To create a timer, the bean invokes one of the createTimer methods of the TimerService Interface. When the bean invokes createTimer, the timer service begins to count down the timer duration.

```
TimerService timerService = context.getTimerService();
Timer timer = timerService.createTimer(intervalDuration,"created timer");
```

In the example, createTimer is invoked in a business method, which is called by a client. An entity bean can also create a timer in a business method. If you want to create a timer for each instance of an entity bean, you can code the createTimer call in the bean's ejbCreate method. Timers are persistent. If the server is shut down (or even crashes), timers are saved and will become active again when the server is restarted. If a timer expires while the server is down, the container will call ejbTimeout when the server is restarted.

The Date and long parameters of the createTimer methods represent time with the resolution of milliseconds. However, because the timer service is not intended for real- time applications, a callback to ejbTimeout might not occur with millisecond precision. The timer service is for business applications, which typically measure time in hours, days, or longer durations.

Cancelling and Saving Timers

- When a single-event timer expires, the EJB container calls ejbTimeout and then cancels the timer.
- When an entity bean instance is removed, the container cancels the timers associated with the instance.
- When the bean invokes the cancel method of the Timer interface, the container cancels the timer.

If a method is invoked on a canceled timer, the container throws the javax.ejb.NoSuchObjectLocalException.

To save a Timer object for future reference, invoke its getHandle method and store the TimerHandle object in a database. (A TimerHandle object is serializable.) To reinstantiate the Timer object, retrieve the handle from the database and invoke getTimer on the handle. A TimerHandle object cannot be passed as an argument of a method defined in a remote or web service interface. In other words, remote clients and web service clients cannot access a bean's TimerHandle object. Local clients, however, do not have this restriction.

Introduction of Hibernate

Hibernate is an Object-Relational Mapping(ORM) solution for JAVA and it raised as an open source persistent framework created by Gavin King in 2001. It is a powerful, high performance Object-Relational Persistence and Query service for any Java Application.

Hibernate maps Java classes to database tables and from Java data types to SQL data types and relieve the developer from 95% of common data persistence related programming tasks.

Hibernate sits between traditional Java objects and database server to handle all the work in persisting those objects based on the appropriate O/R mechanisms and patterns.

Needs of Hibernate

- Hibernate takes care of mapping Java classes to database tables using XML files and without writing any line of code.
- Provides simple APIs for storing and retrieving Java objects directly to and from the database.
- If there is change in Database or in any table then the only need to change XML file properties.
- Abstract away the unfamiliar SQL types and provide us to work around familiar Java Objects.
- Hibernate does not require an application server to operate.
- Manipulates Complex associations of objects of your database.
- Minimize database access with smart fetching strategies.
- Provides simple querying of data.

Hibernate Advantages

- Hibernate supports Inheritance, Associations, Collections
- In hibernate if we save the derived class object, then its base class object will
- also be stored into the database, it means hibernate supporting inheritance
- Hibernate supports relationships like One-To-Many,One-To-One, Many-To-Many- to-Many, Many-To-One
- This will also supports collections like List,Set,Map (Only new collections)
- In jdbc all exceptions are checked exceptions, so we must write code in try, catch and throws, but in hibernate we only have Un-checked exceptions, so no need to write try, catch, or no need to write throws. Actually in hibernate we have the translator which converts checked to Un-checked.
- Hibernate has capability to generate primary keys automatically while we are storing the records into database
- Hibernate has its own query language, i.e hibernate query language which is database independent
- So if we change the database, then also our application will works as HQL is database independent
- HQL contains database independent commands
- While we are inserting any record, if we don't have any particular table in the
- database, JDBC will rises an error like "View not exist", and throws exception, but in case of hibernate, if it not found any table in the database this will create the table for us.
- Hibernate supports caching mechanism by this, the number of round trips between an application and the database will be reduced, by using this caching technique an application performance will be increased automatically.
- Hibernate supports annotations, apart from XML
- Hibernate provided Dialect classes, so we no need to write sql queries in
- hibernate, instead we use the methods provided by that API.
- Getting pagination in hibernate is quite simple.

Hibernate Disadvantages

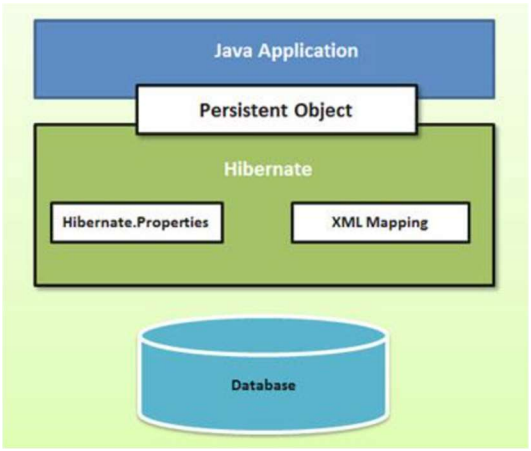
- There is one major disadvantage, which was boilerplate code issue, actually we need to write same code in several files in the same application.

Supported Databases

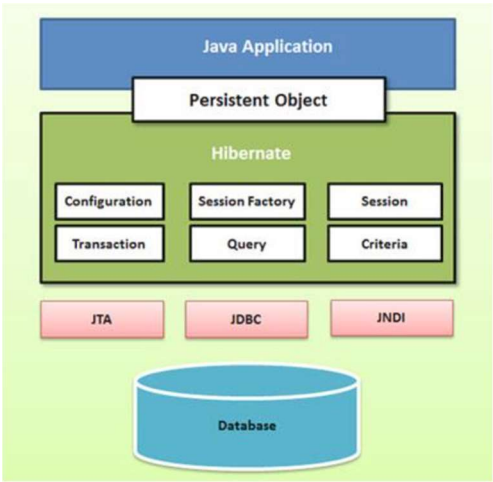
- Hibernate supports almost all the major RDBMS. Following is list of few of the database engines supported by Hibernate.
- HSQL Database Engine DB2/NT
- MySQL
- PostgreSQL
- FrontBase
- Oracle
- Microsoft SQL Server Database Sybase SQL Server
- Informix Dynamic Server

Hibernate Architecture

The Hibernate architecture is layered to keep you isolated from having to know the underlying APIs. Hibernate makes use of the database and configuration data to provide persistence services (and persistent objects) to the application.



Following is a very high level view of the Hibernate Application Architecture.



Following is a detailed view of the Hibernate Application Architecture with few important core classes. Hibernate uses various existing Java APIs, like JDBC, Java Transaction API (JTA), and Java Naming and Directory Interface (JNDI). JDBC provides a rudimentary level of abstraction of functionality common to relational databases, allowing almost any database with a JDBC driver to be supported by Hibernate. JNDI and JTA allow Hibernate to be integrated with J2EE application servers. Following section gives brief description of each of the class objects involved in Hibernate Application Architecture.

Configuration Object

The Configuration object is the first Hibernate object you create in any Hibernate application and usually created only once during application initialization. It represents a configuration or properties file required by the Hibernate. The Configuration object provides two keys 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 Setup: This component creates the connection between the Java classes and database tables.
SessionFactory Object: Configuration object is used to create a SessionFactory object which inturn configures Hibernate for the application using the supplied configuration file and allows for a Session object to be instantiated. The SessionFactory is a thread safe object and used by all the threads of an application. The SessionFactory is heavyweight object so usually it is created during application start up

and kept for later use. You would need one SessionFactory object per database using a separate configuration file. So if you are using multiple databases then you would have to create multiple SessionFactory objects.

Session Object:
A Session is used to get a physical connection with a database. The Session object is lightweight and designed to be instantiated each time an interaction is needed with the database. Persistent objects are saved and retrieved through a Session object.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 them as needed.

Transaction Object:
A Transaction represents a unit of work with the database and most of the RDBMS supports transaction functionality. Transactions in Hibernate are handled by an underlying transaction manager and transaction (from JDBC or JTA).This is an optional object and Hibernate applications may choose not to use this interface, instead managing transactions in their own application code.

Query Object:
Query objects use SQL or Hibernate Query Language (HQL) string to retrieve data from the database and create objects. A Query instance is used to bind query parameters, limit the number of results returned by the query, and finally to execute the query.

Criteria Object:
Criteria object are used to create and execute object oriented criteria queries to retrieve objects.

Hibernate Environment Setup
This will explain how to install Hibernate and other associated packages to prepare a develop environment for the Hibernate applications. We will work with MySQL database to experiment with Hibernate examples, so make sure you already have setup for MySQL database.

Downloading Hibernate
It is assumed that you already have latest version of Java is installed on your machine. Following are the simple steps to download and install Hibernate on your machine. Make a choice whether you want to install Hibernate on Windows, or UNIX and then proceed to the next step to download .zip file for windows and .tz file for UNIX.
Download the latest version of Hibernate from http://www.hibernate.org/downloads.

Installing Hibernate
Once you downloaded and unzipped the latest version of the Hibernate Installation file, you need to perform following two simple steps. Make sure you are setting your CLASSPATH variable properly otherwise you will face problem while compiling your application.
Now copy all the library files from /lib into your CLASSPATH, and change your classpath variable to include all the JARs:
Finally copy hibernate3.jar file into your CLASSPATH. This file lies in the root directory of the installation and is the primary JAR that Hibernate needs to do its work.

Hibernate Prerequisites

Following is the list of the packages/libraries required by Hibernate and you should install them before starting with Hibernate. To install these packages you would have to copy library files from /lib into your CLASSPATH, and change your CLASSPATH variable accordingly.
dom4j - XML parsing www.dom4j.org/
Xalan - XSLT Processor http://xml.apache.org/xalan-j/
Xerces - The Xerces Java Parser http://xml.apache.org/xerces-j/
cglib - Appropriate changes to Java classes at runtime http://cglib.sourceforge.net/
log4j - Logging Faremwork http://logging.apache.org/log4j
Commons - Logging, Email etc. http://jakarta.apache.org/commons
SLF4J - Logging Facade for Java http://www.slf4j.org

Jar files of Hibernate

Files we need to download to work with hibernate framework, and how to install.
Working with the framework software is nothing but, adding the .jar(s) files provided by that framework to our java application. Each framework software is not an installable software, it means we do not contain any setup.exe When we download any framework software, we will get a 'zip' file and we need to unzip it, to get the jar files required, actually all framework software will follow same common principles like.

- Framework software will be in the form of a set of jar files, where one jar file acts as main (We can call this file as core) and remaining will acts as dependent jar files.
- Each Framework software contain at least one configuration xml file, but multiple configuration files also allowed.
- In this case, in order to setup the Hibernate framework environment into a java application, the configuration file is the first one to be loaded into a java application, will see about this in later sessions.

Where to download Hibernate .jar(s) files

we can download jars related to hibernate at http://sourceforge.net/projects/hibernate/files/hibernate3

From the above URL choose hibernate 3.2.2-ga.zip, as we are in initial stage this version will be better. Unzip it, and now you can find some jar files in the lib folder, actually we doesn't require all the jar files, out of them just select the following jar files.

Anttr-2.7.6.jar
asm.jar
asm-attrs.jar
cglib-2.1.3.jar
commons-collections-2.1.1.jar
commons-logging-1.0.4.jar
ehcash.jar
dom4j-1.6.1.jar
hibernate3.jar

jta.jar
log4j-1.2.3.jar

These are the main jar files to run hibernate related programming and among all the jars hibernate3.jar is the main file, but for annotation we need to add 4 – 6 other jar files.
Remember: along with the hibernate jars we must include one more jar file, which is nothing but related to our database, this is depending on your database. So finally we need total of 12 jar files to run the hibernate related program.

Hibernate Configuration

Hibernate requires to know in advance where to find the mapping information that defines how your Java classes relate to the database tables. Hibernate also requires a set of configuration settings related to database and other related parameters. All such information is usually supplied as a standard Java properties file called hibernate.properties, or as an XML file named hibernate.cfg.xml.
I will consider XML formatted filehibernate.cfg.xmlto specify required Hibernate properties in my examples. Most of the properties take their default values and it is not required to specify them in the property file unless it is really required. This file is kept in the root directory of your application's classpath.

Hibernate Properties:
Following is the list of important properties you would require to configure for a databases in a standalone situation:

hibernate.dialect: This property makes Hibernate generate the appropriate SQL for the chosen database.
hibernate.connection.driver_class: The JDBC driver class.
hibernate.connection.url: The JDBC URL to the database instance.
hibernate.connection.username: The database username.
hibernate.connection.password: The database password.
hibernate.connection.pool_size: Limits the number of connections waiting in the Hibernate database connection pool.
hibernate.connection.autocommit: Allows autocommit mode to be used for the JDBC connection.

hibernate.connection.datasource: The JNDI name defined in the application server context you are using for the application.
hibernate.jndi.class: The InitialContext class for JNDI.
hibernate.jndi.<JNDIpropertyname>: Passes any JNDI property you like to the JNDI InitialContext.
hibernate.jndi.url: Provides the URL for JNDI.
hibernate.connection.username: The database username.
hibernate.connection.password: The database password.

MySQL is one of the most popular open-source database systems available today. Let us create hibernate.cfg.xmlconfiguration file and place it in the root of your application's classpath. You would have to make sure that you have testdb database available in your MySQL database and you have a user test available to access the database.

The XML configuration file must conform to the Hibernate 3 Configuration DTD, which is available from http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration SYSTEM "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration> <session-factory>
<property name="hibernate.dialect">
org.hibernate.dialect.MySQLDialect
</property>
<property name="hibernate.connection.driver_class">
com.mysql.jdbc.Driver
</property>
<!-- Assume test is the database name -->
<property name="hibernate.connection.url">
jdbc:mysql://localhost/test
</property>
<property name="hibernate.connection.username">
root
</property>
<property name="hibernate.connection.password">
root123
</property>
<!-- List of XML mapping files -->
<mapping resource="Employee.hbm.xml"/> </session-factory>
</hibernate-configuration>
```

The above configuration file includes <mapping> tags which are related to hibernate- mapping file and we will see in next chapter what exactly is a hibernate mapping file and how and why do we use it. Following is the list of various important databases dialect property type:

Database	Dialect Property
DB2	org.hibernate.dialect.DB2Dialect
HSQLDB	org.hibernate.dialect.HSQLDialect
HypersonicSQL	org.hibernate.dialect.HSQLDialect
Informix	org.hibernate.dialect.InformixDialect
Ingres	org.hibernate.dialect.IngresDialect
Interbase	org.hibernate.dialect.InterbaseDialect
Microsoft SQL Server 2000	org.hibernate.dialect.SQLServerDialect
Microsoft SQL Server 2005	org.hibernate.dialect.SQLServer2005Dialect
Microsoft SQL Server 2008	org.hibernate.dialect.SQLServer2008Dialect
MySQL	org.hibernate.dialect.MySQLDialect
Oracle (any version)	org.hibernate.dialect.OracleDialect
Oracle 11g	org.hibernate.dialect.Oracle10gDialect
Oracle 10g	org.hibernate.dialect.Oracle10gDialect
Oracle 9i	org.hibernate.dialect.Oracle9iDialect
PostgreSQL	org.hibernate.dialect.PostgreSQLDialect
Progress	org.hibernate.dialect.ProgressDialect
SAP DB	org.hibernate.dialect.SAPDBDialect
Sybase	org.hibernate.dialect.SybaseDialect
Sybase Anywhere	org.hibernate.dialect.SybaseAnywhereDialect

Let us try an example of using Hibernate to provide Java persistence in a standalone application. We will go through different steps involved in creating Java Application using Hibernate technology.

Create POJO Classes

The first step in creating an application is to build the Java POJO class or classes, depending on the application that will be persisted to the database. Let us consider our Employee class with getXXX andsetXXX methods to make it JavaBeans compliant class.
A POJO (Plain Old Java Object) is a Java object that doesn't extend or implement some specialized classes and interfaces respectively required by the EJB framework. All normal Java objects are POJO.
When you design a classs to be persisted by Hibernate, it's important to provide JavaBeans compliant code as well as one attribute which would work as index like id attribute in the Employee class.

```
public class Employee {
private int id;
private String firstName;
private String lastName;
private int salary;
public Employee() {}
public Employee(String fname, String lname, int salary) {
this.firstName = fname; this.lastName = lname;
this.salary = salary; }
public int getId() { return id;
}
public void setId( int id )
{
this.id = id;
}
 public String getFirstName()
{ return firstName;
}
public void setFirstName( String first_name )
{
this.firstName = first_name; }
public String getLastName()
{ return lastName;
}
public void setLastName( String last_name )
{
this.lastName = last_name;
}
public int getSalary()
{ return salary;
}
 public void setSalary( int salary )
{ this.salary = salary;
} }
```

Create Database Tables

Second step would be creating tables in your database. There would be one table corresponding to each object you are willing to provide persistence. Consider above objects need to be stored and retrieved into the following RDBMS table:

```
create table EMPLOYEE (
id INT NOT NULL auto_increment,
first_name VARCHAR(20) default NULL, last_name VARCHAR(20) default NULL, salary INT default NULL,
PRIMARY KEY (id)
);
```

Create Mapping Configuration File

This step is to create a mapping file that instructs Hibernate how to map the defined class or classes to the database tables.

```
<?xml version="1.0" encoding="utf-8"?> <!DOCTYPE hibernate-mapping PUBLIC
"-//Hibernate/Hibernate Mapping DTD//EN" "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
<class name="Employee" table="EMPLOYEE">
<meta attribute="class-description"> This class contains the employee detail.
</meta>
<id name="id" type="int" column="id">
<generator class="native"/> </id>
<property name="firstName" column="first_name" type="string"/> <property name="lastName"
column="last_name" type="string"/> <property name="salary" column="salary" type="int"/>
</class> </hibernate-mapping>
```

You should save the mapping document in a file with the format <classname>.hbm.xml. We saved our mapping document in the file Employee.hbm.xml. Let us see little detail
about the mapping document:

- The mapping document is an XML document having <hibernate-mapping> as the root element which contains all the <class> elements.
- The <class> elements are used to define specific mappings from a Java classes to the database tables. The Java class name is specified using the name attribute of the class element and the database table name is specified using the table attribute.
- The<meta>element is optional element and can be used to create the class description.
- The <id> element maps the unique ID attribute in class to the primary key of the database table. The name attribute of the id element refers to the property in the class and the column attribute refers to the column in the database table.
- The type attribute holds the hibernate mapping type, this mapping types will convert from Java to SQL data type.
- " The <generator> element within the id element is used to automatically generate the primary key values. Set theclassattribute of the generator element is set to native to let hibernate pick up either identity, sequence or hilo algorithm to create primary key depending upon the capabilities of the underlying database.
- " The <property> element is used to map a Java class property to a column in the database table. The name attribute of the element refers to the property in the class and the column attribute

refers to the column in the database table. The type attribute holds the hibernate mapping type, this mapping types will convert from Java to SQL data type.

Create Application Class

Finally, we will create our application class with the main () method to run the application. We will use this application to save few Employees' records and then we will apply CRUD operations on those records.

Compilation and Execution

Here are the steps to compile and run the above mentioned application. Make sure you have set PATH and CLASSPATH appropriately before proceeding for the compilation and execution.

Annotation

Hibernate annotations is the newest way to define mappings without a use of XML file. You can use annotations in addition to or as a replacement of XML mapping metadata. Hibernate Annotations is the powerful way to provide the metadata for the Object and Relational Table mapping. All the metadata is clubbed into the POJO java file along with the code this helps the user to understand the table structure and POJO simultaneously during the development.

Environment Setup for Hibernate Annotation

First of all you would have to make sure that you are using JDK 5.0 otherwise you need to upgrade your JDK to JDK 5.0 to take advantage of the native support for annotations. Second, you will need to install the Hibernate 3.x annotations distribution package, available from the sourceforge: (Download Hibernate Annotation) and copy hibernate- annotations.jar, lib/hibernate-comons-annotations.jar and lib/ejb3-persistence.jar from the Hibernate Annotations distribution to your CLASSPATH.

Annotated Class Example

As I mentioned above while working with Hibernate Annotation all the metadata is clubbed into the POJO java file along with the code this helps the user to understand the table structure and POJO simultaneously during the development.

Consider we are going to use following EMPLOYEE table to store our objects:

```
create table EMPLOYEE (
id INT NOT NULL auto_increment, first_name VARCHAR(20) default NULL, last_name VARCHAR(20) default NULL, salary INT default NULL,
PRIMARY KEY (id) );
```

Following is the mapping of Employee class with annotations to map objects with the defined EMPLOYEE table.

```
import javax.persistence.*;
@Entity
@Table(name = "EMPLOYEE")
public class Employee {
@Id @GeneratedValue
@Column(name = "id")
private int id;
@Column(name = "first_name")
private String firstName;
```

```
@Column(name = "last_name")
private String lastName;
@Column(name = "salary")
private int salary;

public Employee() {} public int getId() { return id;
}
public void setId( int id ) { this.id = id;
}
public String getFirstName() { return firstName;
}
public void setFirstName( String first_name ) { this.firstName = first_name;
}
public String getLastName() {
return lastName;
}
public void setLastName( String last_name ) { this.lastName = last_name;
}
public int getSalary() {
return salary;
}
public void setSalary( int salary ) {
this.salary = salary;
}
}
```

Hibernate detects that the @Id annotation is on a field and assumes that it should access properties on an object directly through fields at runtime. If you placed the @Id annotation on the getId () method, you would enable access to properties through getter and setter methods by default. Hence, all other annotations are also placed on either fields or getter methods, following the selected strategy. Following section will explain the annotations used in the above class.

@Entity Annotation

The EJB 3 standard annotations are contained in the javax.persistence package, so we import this package as the first step. Second we used the @Entity annotation to the Employee class which marks this class as an entity bean, so it must have a no-argument constructor that is visible with at least protected scope.

@Table Annotation

The @Table annotation allows you to specify the details of the table that will be used to persist the entity in the database.The @Table annotation provides four attributes, allowing you to override the name of the table, its catalogue, and its schema, and enforce unique constraints on columns in the table. For now we are using just table name which is EMPLOYEE.

@Id and @GeneratedValue Annotations

Each entity bean will have a primary key, which you annotate on the class with the @Id annotation. The primary key can be a single field or a combination of multiple fields depending on your table structure. By default, the @Id annotation will automatically determine the most appropriate primary key generation strategy to be used but you can override this by applying the @GeneratedValue annotation which takes two parameters strategy and generator which I'm not going to discuss here, so let us use only default the default key generation strategy. Letting Hibernate determine which generator type to use makes your code portable between different databases.

@Column Annotation

The @Column annotation is used to specify the details of the column to which a field or property will be mapped. You can use column annotation with the following most commonly used attributes:

- name attribute permits the name of the column to be explicitly specified.
- length attribute permits the size of the column used to map a value particularly
- for a String value.
- nullable attribute permits the column to be marked NOT NULL when the schema is generated.
- unique attribute permits the column to be marked as containing only unique values.

Inheritance Annotations

We can map the inheritance hierarchy classes with the table of the database. There are three inheritance mapping strategies defined in the hibernate:

Table per Hierarchy

In table per hierarchy mapping, single table is required to map the whole hierarchy, an extra column (known as discriminator column) is added to identify the class. But nullable values are stored in the table.

Table per Concrete class

In case of table per concrete class, tables are created as per class. But duplicate column is added in subclass tables.

Table per Subclass

In this strategy, tables are created as per class but related by foreign key. So there are no duplicate columns.

Hibernate Sessions

A Session is used to get a physical connection with a database. The Session object is lightweight and designed to be instantiated each time an interaction is needed with the database. Persistent objects are saved and retrieved through a Session object.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 them as needed. The main function of the Session is to offer create, read and delete operations for instances of mapped entity classes. Instances may exist in one of the following three states at a given point in time:

transient: A new instance of a a persistent class which is not associated with a Session and has no representation in the database and no identifier value is considered transient by Hibernate.
persistent: You can make a transient instance persistent by associating it with a Session. A persistent instance has a representation in the database, an identifier value and is associated with a Session.

detached: Once we close the Hibernate Session, the persistent instance will become a detached instance.

Session Interface Methods

There are number of methods provided by the Session interface but I'm going to list down few important methods only, which we will use in this tutorial. You can check Hibernate documentation for a complete list of methods associated with Session and SessionFactory

No	Session Methods and Description
1	**Transaction beginTransaction()** Begin a unit of work and return the associated Transaction object.
2	**void cancelQuery()** Cancel the execution of the current query.
3	**void clear()** Completely clear the session.
4	**Connection close()** End the session by releasing the JDBC connection and cleaning up.
5	**Criteria createCriteria(Class persistentClass)** Create a new Criteria instance, for the given entity class, or a superclass of an entity class.
6	**Criteria createCriteria(String entityName)** Create a new Criteria instance, for the given entity name.
7	**Serializable getIdentifier(Object object)** Return the identifier value of the given entity as associated with this session.
8	**Query createFilter(Object collection, String queryString)** Create a new instance of Query for the given collection and filter string.
9	**Query createQuery(String queryString)** Create a new instance of Query for the given HQL query string.
10	**SQLQuery createSQLQuery(String queryString)** Create a new instance of SQLQuery for the given SQL query string.
11	**void delete(Object object)** Remove a persistent instance from the datastore.
12	**void delete(String entityName, Object object)** Remove a persistent instance from the datastore.
13	**Session get(String entityName, Serializable id)** Return the persistent instance of the given named entity with the given identifier, or null if there is no such persistent instance.
14	**SessionFactory getSessionFactory()** Get the session factory which created this session.
15	**void refresh(Object object)** Re-read the state of the given instance

	from the underlying database.
16	**Transaction getTransaction()** Get the Transaction instance associated with this session.
17	**boolean isConnected()** Check if the session is currently connected.
18	**boolean isDirty()** Does this session contain any changes which must be synchronized with the database?
19	**boolean isOpen()** Check if the session is still open.
20	**Serializable save(Object object)** Persist the given transient instance, first assigning a generated identifier.
21	**void saveOrUpdate(Object object)** Either save(Object) or update(Object) the given instance.
22	**void update(Object object)** Update the persistent instance with the identifier of the given detached instance.
23	**void update(String entityName, Object object)** Update the persistent instance with the identifier of the given detached instance.

Hibernate Inheritance

Compared to JDBC we have one main advantage in hibernate, which is hibernate inheritance. Suppose if we have base and derived classes, now if we save derived(sub) class object, base class object will also be stored into the database.But the thing is we must specify in what table we need to save which object data.

Note: We can also called this Hibernate Inheritance Mapping as Hibernate Hierarchy