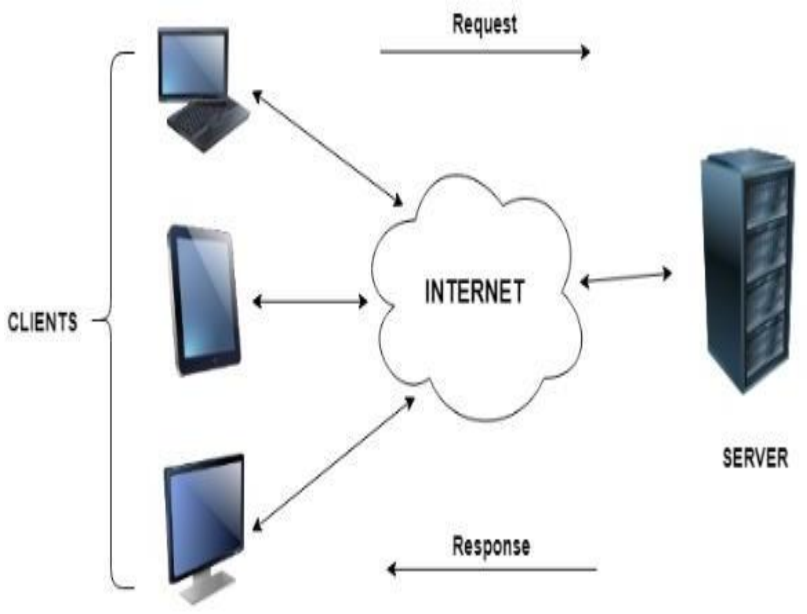
Advanced Java Programming (CIE-306T)

Unit -4

- The Roles of Client and Server, Remote Method Invocations, Setup for Remote Method Invocation, Parameter Passing in Remote Methods, Introduction of HB, HB Architecture

The Roles of Client and Server

- In client server computing, the clients requests a resource and the server provides that resource. A server may serve multiple clients at the same time while a client is in contact with only one server. Both the client and server usually communicate via a computer network but sometimes they may reside in the same system.
- An illustration of the client server system is given as follows –



Characteristics of Client Server Computing

The salient points for client server computing are as follows:

- The client server computing works with a system of request and response. The client sends a request to the server and the server responds with the desired information.
- The client and server should follow a common communication protocol so they can easily interact with each other. All the communication protocols are available at the application layer.
- A server can only accommodate a limited number of client requests at a time. So it uses a system based to priority to respond to the requests.
- Denial of Service attacks hindera servers ability to respond to authentic client requests by inundating it with false requests.
- An example of a client server computing system is a web server. It returns the web pages to the clients that requested them.

Difference between Client Server Computing and Peer to Peer Computing

The major differences between client server computing and peer to peer computing are as follows:

- In client server computing, a server is a central node that services many client nodes. On the other hand, in a peer to peer system, the nodes collectively use their resources and communicate with each other.
- In client server computing the server is the one that communicates with the other nodes. In peer to peer to computing, all the nodes are equal and share data with each other directly.
- Client Server computing is believed to be a subcategory of the peer to peer computing.

Advantages of Client Server Computing

The different advantages of client server computing are –

- All the required data is concentrated in a single place i.e. the server. So it is easy to protect the data and provide authorisation and authentication.
- The server need not be located physically close to the clients. Yet the data can be accessed efficiently.
- It is easy to replace, upgrade or relocate the nodes in the client server model because all the nodes are independent and request data only from the server.
- All the nodes i.e clients and server may not be build on similar platforms yet they can easily facilitate the transfer of data.

Disadvantages of Client Server Computing

The different disadvantages of client server computing are –

- If all the clients simultaneously request data from the server, it may get overloaded.This may lead to congestion in the network.
- If the server fails for any reason, then none of the requests of the clients can be fulfilled. This leads of failure of the client server network.
- The cost of setting and maintaining a client server model are quite high.

Remote Method Invocation

- Remote Method Invocation (RMI) is Java's implementation of object-to-object communication among Java objects to realize a distributed computing model.
- RMI allows us to distribute our objects on various machines, and invoke methods on the objects located on remote sites.
- **RMI** is an API that provides a mechanism to create distributed application in java. The RMI allows an object to invoke methods on an object running in another JVM.
- The RMI provides remote communication between the applications using two objects *stub* and *skeleton*.

Understanding stub and skeleton

- RMI uses stub and skeleton object for communication with the remote object.
- A **remote object** is an object whose method can be invoked from another JVM. Let's understand the stub and skeleton objects:

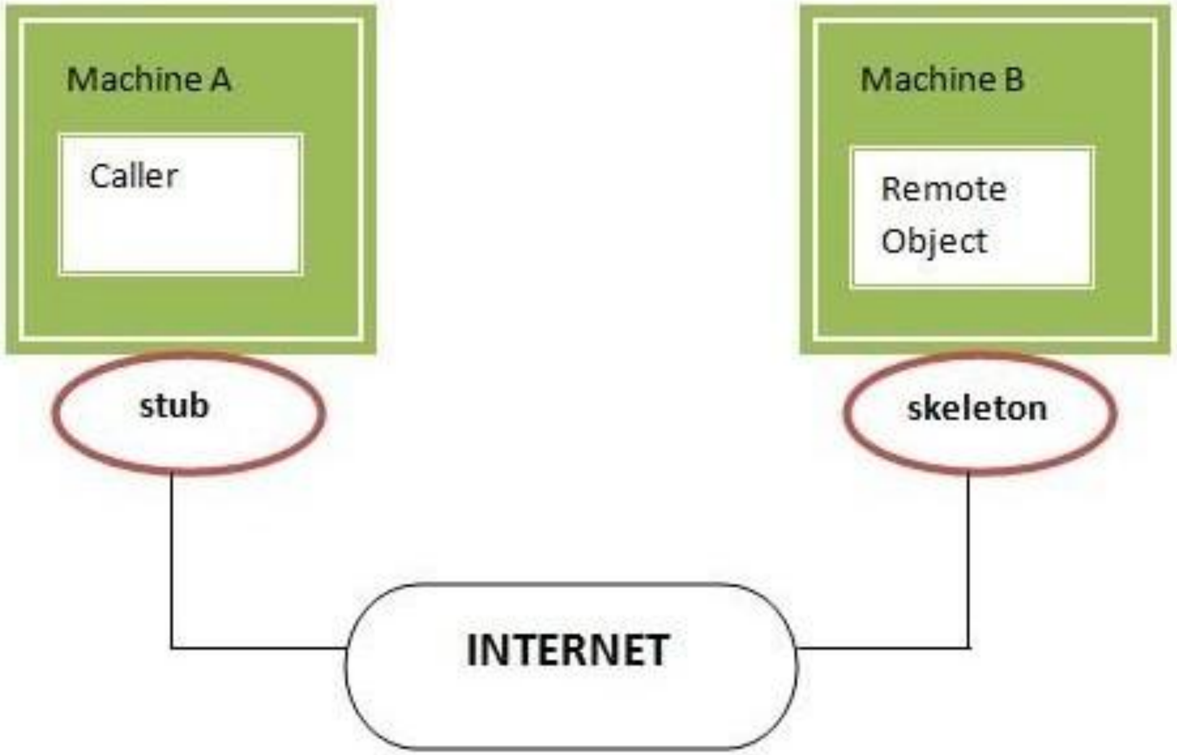
stub

- The stub is an object, acts as a gateway for the client side. All the outgoing requests are routed through it. It resides at the client side and represents the remote object.When the caller invokes method on the stub object, it does the following tasks:
- It initiates a connection with remote Virtual Machine (JVM),
- It writes and transmits (marshals) the parameters to the remote Virtual Machine (JVM),
- It waits for the result
- It reads (unmarshals) the return value or exception, and
- It finally, returns the value to the caller.

skeleton

- The skeleton is an object, acts as a gateway for the server side object. All the incoming requests are routed through it.When the skeleton receives the incoming request, it does the following tasks:
- It reads the parameter for the remote method
- It invokes the method on the actual remote object, and
- It writes and transmits (marshals) the result to the caller.

In the Java 2 SDK, an stub protocol was introduced that eliminates the need for skeletons.

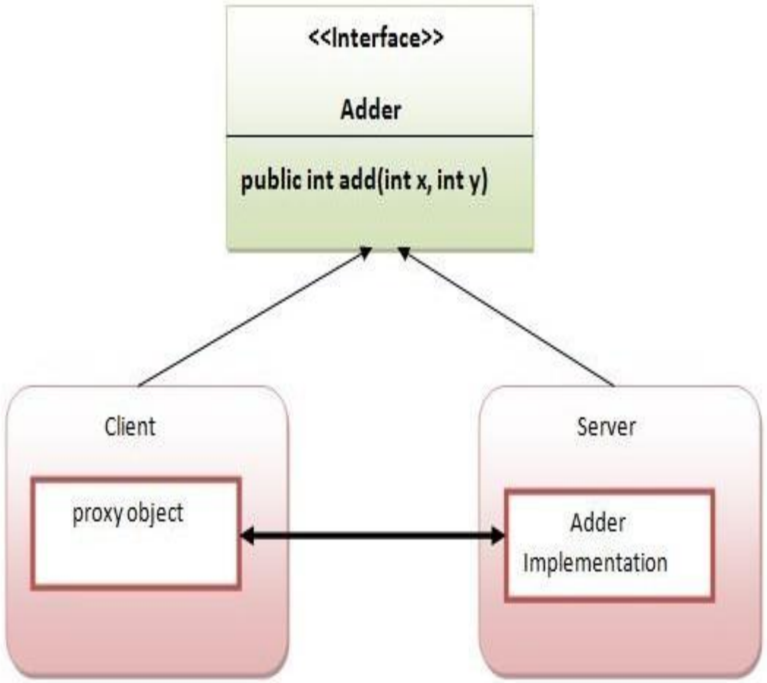


Understanding requirements for the distributed applications

- If any application performs these tasks, it can be distributed application.
- .The application need to locate the remote method
- It need to provide the communication with the remote objects, and
- The application need to load the class definitions for the objects.
- The RMI application have all these features, so it is called the distributed application

RMI Example

- In this example, we have followed all the 6 steps to create and run the rmi application.The client application need only two files, remote interface and client application. In the rmi application, both client and server interacts with the remote interface. The client application invokes methods on the proxy object, RMI sends the request to the remote JVM. The return value is sent back to the proxy object and then to the client application.



Java RMI Example

The is given the 6 steps to write the RMI program.

1. Creation of remote interface
2. Provide the implementation of the remote interface
3. Compile the implementation class and create the stub and skeleton objects using the rmic tool
4. Start the registry service by rmi registry tool
5. Write and start the remote application
6. Write and start the client application

1. Define the java remote interface

- First thing to do is create an interface to describe the techniques that can be invoked by distant clients.This interface should extend the Remote interface and throw the RemoteException inside the interface using the method prototype.

```
// Creating a Search interface
import java.rmi.*;
public interface MySearch extends Remote
{
   // Declaring the method prototype
   public String query(String search) throws RemoteException;
```

2. Implement the java remote interface

The next step is to implement the remote interface.To execute the remote interface, the class should extend to the UnicastRemoteObject class java.rmi package. In addition, a default constructor must be formed to transfer the java.rmi.RemoteException from its class parent constructor.

```
// Java program to implement the MySearch interface
import java.rmi.*;
import java.rmi.server.*;
public class SearchQuery extends UnicastRemoteObject
                    implements MySearch
{
   SearchQuery() throws RemoteException
   {
      super();
   }
// Implementation of the query interface
public String query(String search)
                  throws RemoteException
   {
      String result;
      if (search.equals("Reflection in Java"))
         result = "Yes it's found";
      else
         result = "No its not found";
      return result;
   }
}
```

3. Writing Stub and Skeleton objects from the implementation class using rmic

- The rmic instrument is used to invoke the rmi compiler which produces the Stub and Skeleton items. His prototype is the rmic class's name. For the above program, the following command must be executed at the rmicSearchQuery command prompt.

4. Start/Invoke the rmiregistry

- Start the service of the registry by putting the following command on the prompt of start rmiregistry.

5. Write and execute the server application program

- The next step is the development and execution on a separate command prompt of the server application program.
- The server program uses the creation technique of the LocateRegistry class to produce rmiregistry with the passed port number as the argument within the JVM server.
- The Naming class rebind method is used to bind the remote object to the new name.

//Java server application

```
import java.rmi.*;
import java.rmi.registry.*;
public class SearchServer
{
public static void main(String args[])
{
try
{
// Create an object of the interface
// implementation class
Search obj = new SearchQuery();
// rmiregistry within the server
JVM with
// port number 1900
LocateRegistry.createRegistry(19
00);
// Binds the remote object by the
name
Naming.rebind("rmi://localhost:
1900"+
"/test",obj);
}
catch(Exception ae)
{
System.out.println(ae);
}
}
}
```

6. Write and execute the client application program

- The last stage is to generate and implement the client application program on a distinct command prompt. The Naming class lookup method is used to get the Stub object reference.
- To use localhost, the above client and server program runs on the same machine.To access the remote object from another device, the localhost must be replaced with the IP address where the remote object is present.

```
//Java client application
import java.rmi.*;
public class ClientRequest
{
  public static void main(String args[])
  {
    String answer,value="Reflection in Java";
    try
    {
      // lookup method to find reference of remote object
      Search access =
        (Search)Naming.lookup("rmi://localhost:1900"+
                    "/test");
      answer = access.query(value);
      System.out.println("Article on " + value +
                " " + answer+" at online");
    }
    catch(Exception ae)
    {
      System.out.println(ae);
    }  }  }
```

Parameter Passing in Remote Methods

- An argument to, or a return value from, a remote object can be any Java type that is serializable. This includes Java primitive types, remote Java objects, and nonremote Java objects that implement the java.io.Serializable interface. For more details on how to make classes serializable, see the Java "Object Serialization Specification." For applets, if the class of an argument or return value is not available locally, it is loaded dynamically via the AppletClassLoader. For applications, these classes are loaded by the class loader that loaded the application; this is either the default class loader (which uses the local class path) or the RMIClassLoader (which uses the server's codebase).
- Some classes may disallow their being passed (by not being serializable), for example for security reasons. In this case the remote method invocation will fail with an exception.

Passing Nonremote Objects

- A nonremote object, that is passed as a parameter of a remote method invocation or returned as a result of a remote method invocation, is passed by *copy*.That is, when a nonremote object appears in a remote method invocation, the content of the nonremote object is copied before invoking the call on the remote object. By default, only the nonstatic and nontransient fields are copied.
- Similarly, when a nonremote object is returned from a remote method invocation, a new object is created in the calling virtual machine.

Passing Remote Objects

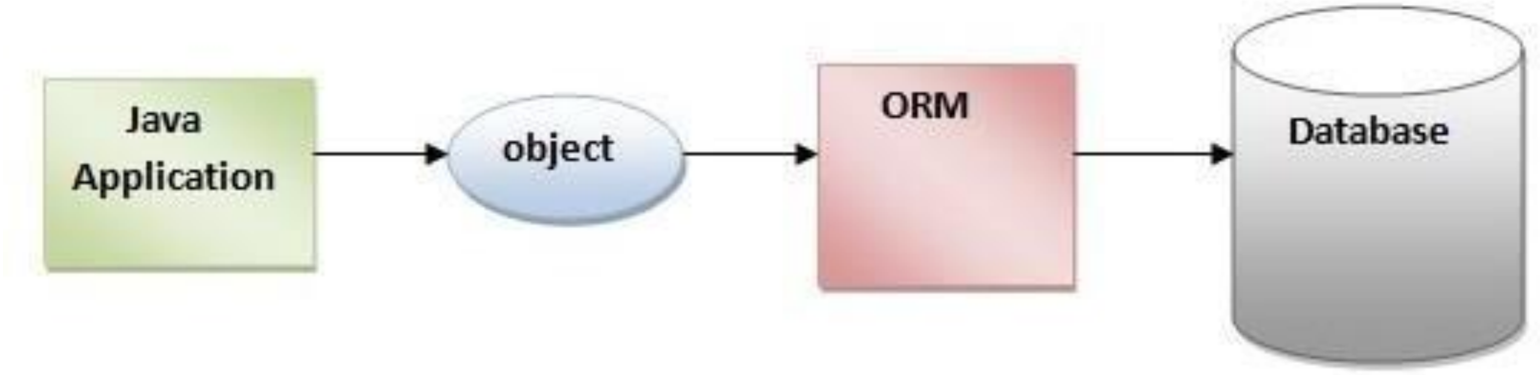
- When passing a remote object as a parameter, the stub for the remote object is passed. A remote object passed as a parameter can only implement remote interfaces.

Exception Handling in Remote Method Invocation

- Since remote methods include java.rmi.RemoteException in their signature, the caller must be prepared to handle those exceptions in addition to other application specific exceptions. When a java.rmi.RemoteException is thrown during a remote method invocation, the client may have little or no information on the outcome of the call -- whether a failure happened before, during, or after the call completed. Therefore, remote interfaces and the calling methods declared in those interfaces should be designed with these failure semantics in mind.

Introduction to HB

- Hibernate is a Java framework that simplifies the development of Java application to interact with the database. It is an open source, lightweight, ORM (Object Relational Mapping) tool. Hibernate implements the specifications of JPA (Java Persistence API) for data persistence.
- An ORM tool simplifies the data creation, data manipulation and data access. It is a programming technique that maps the object to the data stored in the database. The ORM tool internally uses the JDBC API to interact with the database.
- Java Persistence API (JPA) is a Java specification that provides certain functionality and standard to ORM tools.The **javax.persistence** package contains the JPA classes and interfaces.



Advantages of Hibernate Framework

Following are the advantages of hibernate framework:

1) Open Source and Lightweight

Hibernate framework is open source under the LGPL license and lightweight.

2) Fast Performance

The performance of hibernate framework is fast because cache is internally used in hibernate framework. There are two types of cache in hibernate framework first level cache and second level cache. First level cache is enabled by default.

3) Database Independent Query

HQL (Hibernate Query Language) is the object-oriented version of SQL. It generates the database independent queries. So you don't need to write database specific queries. Before Hibernate, if database is changed for the project, we need to change the SQL query as well that leads to the maintenance problem.

4) Automatic Table Creation

Hibernate framework provides the facility to create the tables of the database automatically. So there is no need to create tables in the database manually.

5) Simplifies Complex Join

Fetching data from multiple tables is easy in hibernate framework.

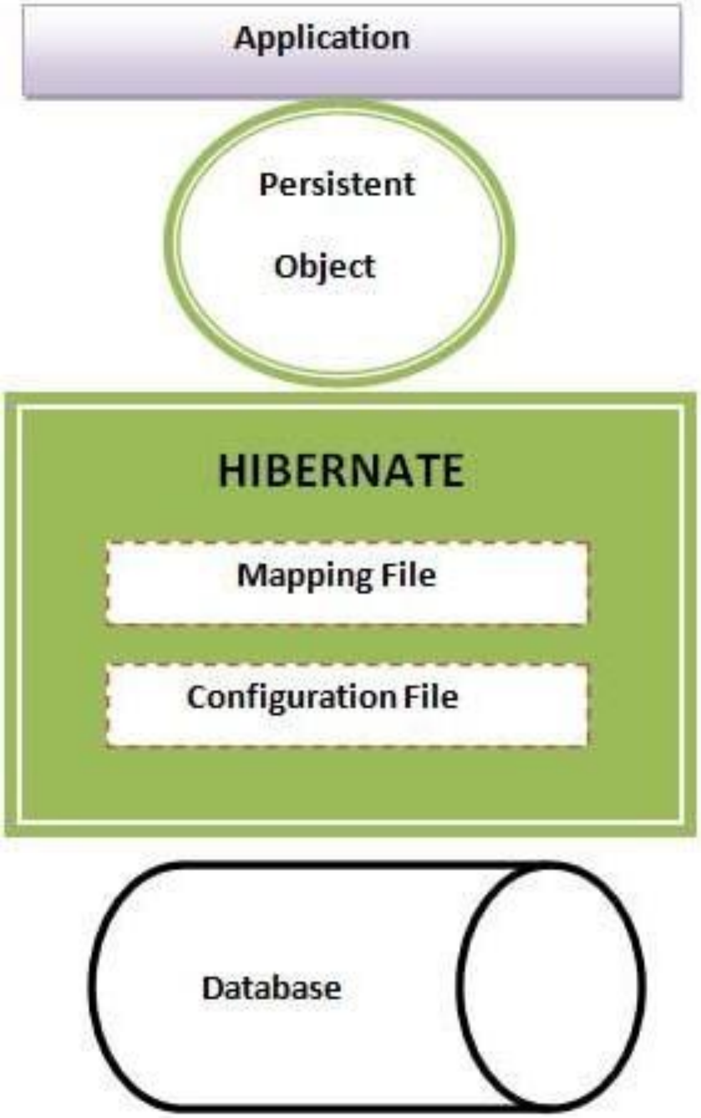
6) Provides Query Statistics and Database Status

Hibernate supports Query cache and provide statistics about query and database status.

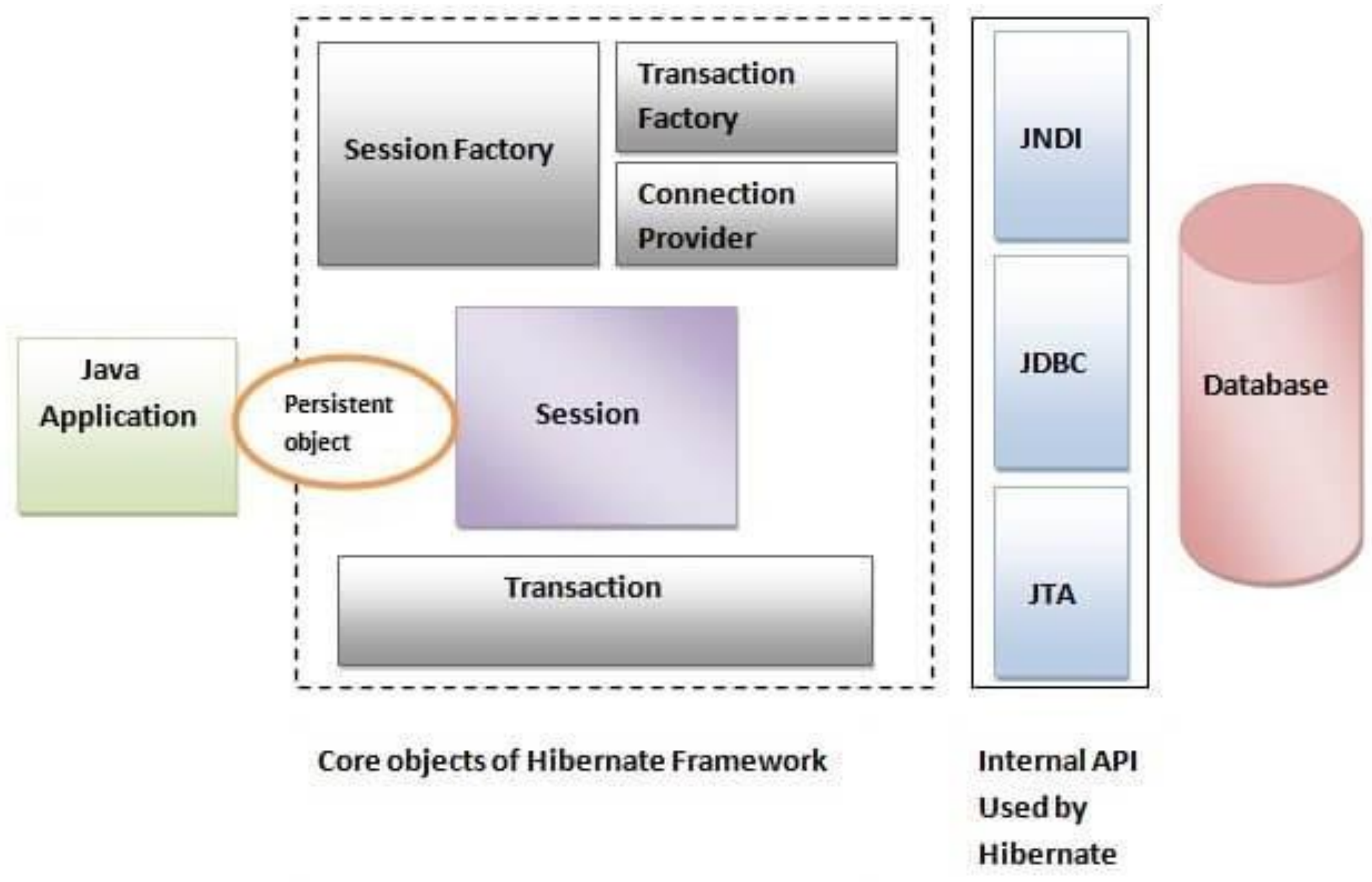
Hibernate Architecture

- The Hibernate architecture includes many objects such as persistent object, session factory, transaction factory, connection factory, session, transaction etc.
- The Hibernate architecture is categorized in four layers.
  1. Java application layer
  2. Hibernate framework layer
  3. Backhand api layer
  4. Database layer
- Hibernate framework uses many objects such as session factory, session, transaction etc. alongwith existing Java API such as JDBC (Java Database Connectivity), JTA (Java Transaction API) and JNDI (Java Naming Directory Interface).

Hibernate Architecture



High level architecture of Hibernate with mapping file and configuration file.



Elements of Hibernate Architecture

- For creating the first hibernate application, we must know the elements of Hibernate architecture. They are as follows:

SessionFactory

- The SessionFactory is a factory of session and client of ConnectionProvider. It holds second level cache (optional) of data.The org.hibernate.SessionFactory interface provides factory method to get the object of Session.

Session

- The session object provides an interface between the application and data stored in the database. It is a short-lived object and wraps the JDBC connection. It is factory of Transaction, Query and Criteria. It holds a first-level cache (mandatory) of data.The org.hibernate.Session interface provides methods to insert, update and delete the object. It also provides factory methods for Transaction, Query and Criteria.

Transaction

- The transaction object specifies the atomic unit of work. It is optional.The org.hibernate.Transaction interface provides methods for transaction management.

ConnectionProvider

- It is a factory of JDBC connections. It abstracts the application from DriverManager or DataSource. It is optional.

TransactionFactory

- It is a factory of Transaction. It is optional.