Java EE

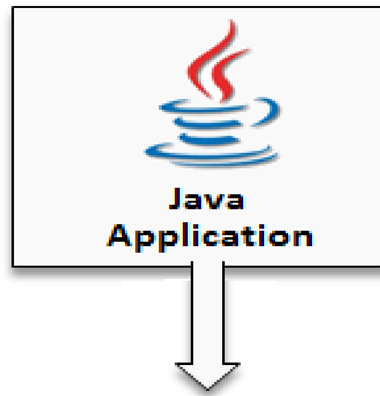
→ Mrs. Soniya Sharma.

Steps to connect to DB

- Import JDBC packages.
  - import java.sql.*;
- Load and register the JDBC driver.
  - DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
  - Class.forName("oracle.jdbc.driver.OracleDriver");
- Open a connection to the database.
  - Connection conn = DriverManager.getConnection(URL, username, passwd); Connection conn = DriverManager.getConnection(URL);
- Create a statement object to perform a query.
  - Statement sql_stmt = conn.createStatement();

Steps to connect to DB

- Execute the statement object and return a query resultset.
  - ResultSet rset = sql_stmt.executeQuery ("SELECT empno, ename, sal, deptno FROM emp ORDER BY ename");
- Process the resultset.
  - while (rset.next())
  - getInt(), getString() ,getFloat()
- Close the resultset and statement objects.
  - rset.close()
  - sql_stmt.close()
- Close the connection.
  - conn.close();



Sitesbay.com

Sitesbay.com

Type 1

JDBC ODBC Bridge Driver

ODBC Driver

Vendar DataBase Library

Type 2

Native API Driver

Vendar DataBase Library

Type 3

Network Protocol Driver

Middleware

Type 4

Thin Driver

DataBase

DataBase

DataBase

DataBase

Driver	Database	Platform
Type - 1	Independent	Dependent.
Type - 2	Dependent	Dependent.
Type - 3	Independent	Independent.
Type - 4	Dependent	Independent.

JDBC-ODBC BRIDGE DRIVER:

This driver connect a java program with a database using Odbc driver. It is install automatically along with JDK software. It is provided by Sun MicroSystem for testing purpose this driver can not be used in real time application. This driver convert JDBC calls into Odbc calls(function) So this is called a **bridge driver**.

Advantage of bridge driver

Easy to use

Can be easily connected to any database.

This driver software is built-in with JDK so no need to install separately.

Disadvantage of bridge driver

It is a slow driver so not used in real time application

It is not a portable driver.

- Jdbc-Native API :
- JDBC API calls are converted into native c/c++ API calls which are unique to the database. For this vendor specific driver must be installed on each client machine.
- **Advantage of Thin driver**
- Native API driver comparatively faster than JDBC-ODBC bridge driver.
- **Disadvantage of Thin driver**
- Native API driver is database dependent and also platform dependent because of Native API.

RequestDispatcher interface

- The **RequestDispatcher** interface defines an object that receives the request from client and dispatches it to the resource(such as servlet, JSP, HTML file). This interface has following two methods:
- **public void forward(ServletRequest request, ServletResponse response)**: It forwards the request from one servlet to another resource (such as servlet, JSP, HTML file).
- **public void include(ServletRequest request, ServletResponse response)**: It includes the content of the resource(such as servlet, JSP, HTML file) in the response.

index.html

```
<form action="loginPage" method="post">
  User Name:<input type="text" name="uname"/><br/>
  Password:<input type="password" name="upass"/><br/>
  <input type="submit" value="SUBMIT"/>
</form>
```

Validation.java

```
import java.io.*;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class Validation extends HttpServlet
{
   public void doPost(HttpServletRequest request,
     HttpServletResponse response)
       throws ServletException, IOException
   {
       response.setContentType("text/html");
       PrintWriter pwriter = response.getWriter();
       String name=request.getParameter("uname");
       String pass=request.getParameter("upass");
       if(name.equals("Chaitanya") &&
          pass.equals("beginnersbook"))
       {
          RequestDispatcher dis=request.getRequestDispatcher("welcome");
          dis.forward(request, response);
       }
       else
       {
          pwriter.print("User name or password is incorrect!");
          RequestDispatcher dis=request.getRequestDispatcher("index.html");
          dis.include(request, response);
       }
   }
}
```

WelcomeUser.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class WelcomeUser extends HttpServlet {

 public void doPost(HttpServletRequest request,
 HttpServletResponse response)
 throws ServletException, IOException
 {

    response.setContentType("text/html");
    PrintWriter pwriter = response.getWriter();

    String name=request.getParameter("uname");
    pwriter.print("Hello "+name+"!");
    pwriter.print(" Welcome to Beginnersbook.com");
 }

}
```

web.xml

```
<web-app>
 <display-name>BeginnersBookDemo</display-name>
 <welcome-file-list>
 <welcome-file>index.html</welcome-file>
 </welcome-file-list>

 <servlet>
 <servlet-name>Login</servlet-name>
 <servlet-class>Validation</servlet-class>
 </servlet>
 <servlet>
 <servlet-name>Welcome</servlet-name>
 <servlet-class>WelcomeUser</servlet-class>
 </servlet>
 <servlet-mapping>
 <servlet-name>Login</servlet-name>
 <url-pattern>/loginPage</url-pattern>
 </servlet-mapping>
 <servlet-mapping>
 <servlet-name>Welcome</servlet-name>
 <url-pattern>/welcome</url-pattern>
 </servlet-mapping>
 <welcome-file-list>
 <welcome-file>index.html</welcome-file>
 </welcome-file-list>
</web-app>
```

Output:

Entering wrong credentials:

Error screen:

Welcome screen on entering correct user name and password:

Cookies

- Cookie is a small piece of information that is persisted between the multiple client requests.
- A cookie has a name, single value and optional attributes such as comment, path , maximum age and version number.
- By default, each request is considered as a new request. In cookies technique, we add cookie with response from the servlet. So cookie is stored in the cache of the browser.
- After that if request is sent by the user, cookie is added with request by default. Thus we recognize user as the old user.

Kinds of Cookies

- **There are two kinds of cookies in servlet.**
  - **Non-persistent Cookie / session cookies**
  - It is valid for single session only. It is removed each time when user closes the browser.
  - Session cookies are stored in memory and are accessible as long as the user is using the web application. Session cookies are lost when user exits the web application.
  - Session cookies are identified by session id and are most commonly used to store details of shopping cart.
  - **Persistent Cookie / permanent cookies**
  - It is valid for multiple sessions. It is not removed each time when user closes the browser.
  - Permanent cookies are used to store long term information such as user preferences and user identification information. These are stored in persistence storage and are not lost when user exits the application. Permanent cookies are lost when they expire.

Advantages and Disadvantages of Cookies

- **Advantages :**
  - Simplest technique of maintaining the state.
  - Cookies are maintained at client side.

- **Disadvantages:**
  - It will not work if cookie is disabled from the browser.
  - Only textual information can be set in cookie object.
  - Some web browsers limit the number of cookies (typically 20 per web server) that can be installed.
  - Cookies cannot identify a particular user. A user can be identified by a combination of user account, browser and computer. So, users who have multiple accounts and use multiple computers/browsers have multiple sets of cookies. Consequently, cookies cannot differentiate between multiple browsers running in a single computer.
  - Intruders can snoop, steal cookies, and attack sessions. This is called *session hijacking.*

Where Cookies are used?

- **Cookies are most commonly used to tract website activity.**
- Cookies are used for **Online shopping.** Online stores uses cookies that record any personal information you enter as well as any items in your electronic shopping cart. So that you don’t need to reenter this information each time you visit the site.
- Servers can use cookies to **provide personalized web pages**. When you select preferences at site that uses this option, the server places the information in a cookie. When you return, the server uses the information in cookie to create a customized page for you.

Creating cookies using servlet

- **Cookie class**
  - **javax.servlet.http.Cookie** class provides the functionality of using cookies. It provides a lot of useful methods for cookies.
  - Cookie()constructs a cookie.
  - Cookie(String name, String value)constructs a cookie with a specified name and value.

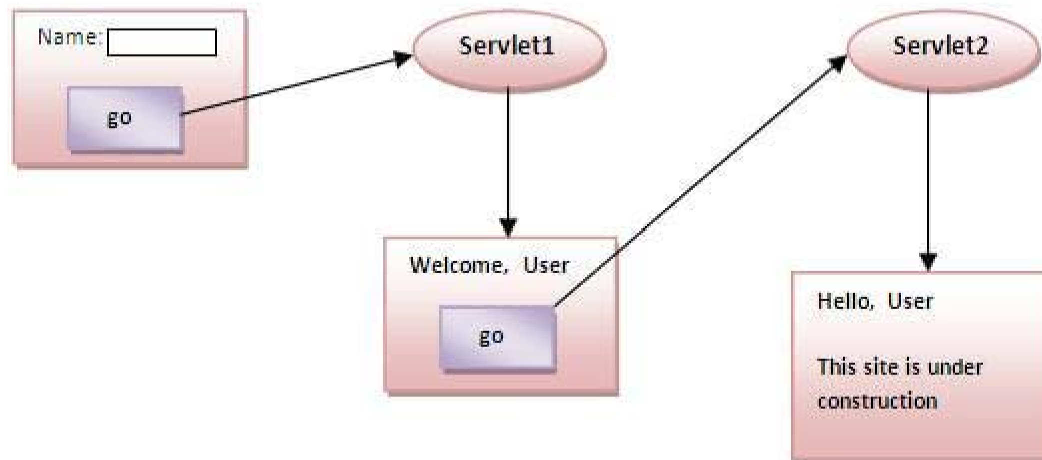
Useful Methods of Cookie class

There are given some commonly used methods of the Cookie class.

Method	Description
public void setMaxAge(int expiry)	Sets the maximum age of the cookie in seconds.
public String getName()	Returns the name of the cookie. The name cannot be changed after creation.
public String getValue()	Returns the value of the cookie.
public void setName(String name)	changes the name of the cookie.
public void setValue(String value)	changes the value of the cookie.

- Other methods required for using Cookies
- **void addCookie(Cookie ck):** method of HttpServletResponse interface is used to add cookie in response object.
- **public Cookie[] getCookies():** method of HttpServletRequest interface is used to return all the cookies from the browser.

Example



index.html

```
<form action="servlet1" method="post">
Name:<input type="text" name="userName"/><br/>
<input type="submit" value="go"/>
</form>
```

FirstServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;


public class FirstServlet extends HttpServlet {

 public void doPost(HttpServletRequest request, HttpServletResponse response){
  try{

  response.setContentType("text/html");
  PrintWriter out = response.getWriter();

  String n=request.getParameter("userName");
  out.print("Welcome "+n);

  Cookie ck=new Cookie("uname",n);//creating cookie object
```

```java
response.setContentType("text/html");
PrintWriter out = response.getWriter();

String n=request.getParameter("userName");
out.print("Welcome "+n);

Cookie ck=new Cookie("uname",n);//creating cookie object
response.addCookie(ck);//adding cookie in the response

//creating submit button
out.print("<form action='servlet2'>");
out.print("<input type='submit' value='go'>");
out.print("</form>");

out.close();

    }catch(Exception e){System.out.println(e);}
  }
}
```

SecondServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class SecondServlet extends HttpServlet {

public void doPost(HttpServletRequest request, HttpServletResponse response){
    try{

    response.setContentType("text/html");
    PrintWriter out = response.getWriter();

    Cookie ck[]=request.getCookies();
    out.print("Hello "+ck[0].getValue());

    out.close();

        }catch(Exception e){System.out.println(e);}
    }


}
```

web.xml

```xml
<web-app>

<servlet>
<servlet-name>s1</servlet-name>
<servlet-class>FirstServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>s1</servlet-name>
<url-pattern>/servlet1</url-pattern>
</servlet-mapping>

<servlet>
<servlet-name>s2</servlet-name>
<servlet-class>SecondServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>s2</servlet-name>
<url-pattern>/servlet2</url-pattern>
</servlet-mapping>

</web-app>
```

Sessions

- HTTP protocol and Web Servers are stateless, what it means is that for web server every request is a new request to process and they can't identify if it's coming from client that has been sending request previously.
- **Session** is a conversional state between client and server and it can consists of multiple request and response between client and server. Since HTTP and Web Server both are stateless, the only way to maintain a session is when some unique information about the session (session id) is passed between server and client in every request and response.

Session Tracking Techniques

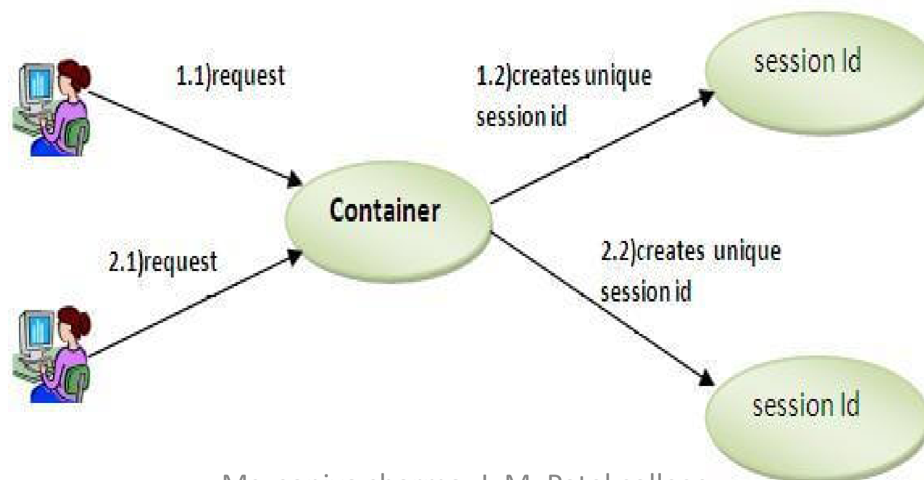
- **To recognize the user** we do session tracking.
- There are four techniques used in Session tracking:
  - **Cookies - Done**
  - **Hidden Form Field**
  - **URL Rewriting**
  - **HttpSession**

Session Tracking Techniques

- **<u>Hidden form field.</u>**
- In case of Hidden Form Field **a hidden (invisible) textfield** is used for maintaining the state of an user.
- In such case, we store the information in the hidden field and get it from another servlet.
- <input type="hidden" name="uname" value="Vimal Jais wal">
- Here, uname is the hidden field name and Vimal Jaiswal is the hidden field value.
- Advantage of Hidden Form Field
  - It will always work whether cookie is disabled or not.
- Disadvantage of Hidden Form Field:
  - It is maintained at server side.
  - Extra form submission is required on each pages.
- Only textual information can be used.

- **URL Rewriting**
- You can append some extra data on the end of each URL that identifies the session, and the server can associate that session identifier with data it has stored about that session.
- For example, with http://tutorialspoint.com/file.htm;sessionid = 12345, the session identifier is attached as sessionid = 12345 which can be accessed at the web server to identify the client.
- Advantage :
  - URL rewriting is a better way to maintain sessions and it works even when browsers don't support cookies.
- Disadvantage:
  - You have to generate every URL dynamically to assign a session ID, even in case of a simple static HTML page.

- **URL Rewriting**
- **HttpSession**
- container creates a session id for each user.The container uses this id to identify the particular user.
- An object of HttpSession can be used to perform two tasks:
- bind objects
- view and manipulate information about a session, such as the session identifier, creation time, and last accessed time.



- Getting the HttpSession object :

The HttpServletRequest interface provides two methods to get the object of HttpSession:

  - **public HttpSession getSession():**Returns current session associated with request, if request does not have a session, creates one.
  - **public HttpSession getSession(boolean create):**Returns the current HttpSession associated with request, if there is no current session and create is true, returns a new session.

- Methods of HttpSession interface
  - **public String getId():**Returns string containing the unique identifier value.
  - **public long getCreationTime():**Returns the time when this session was created
  - **public long getLastAccessedTime():**Returns the last time the client sent request associated with this session
  - **public void invalidate():**Invalidates session, unbinds any objects bound to it.

- index.html

```
<form action="servlet1">
Name:<input type="text" name="userName"/><br/>
<input type="submit" value="go"/>
</form>
```

- FirstServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class FirstServlet extends HttpServlet {
public void doGet(HttpServletRequest request, HttpServletRe
sponse response){
 try{   response.setContentType("text/html");
        PrintWriter out = response.getWriter();
         String n=request.getParameter("userName");
        out.print("Welcome "+n);
         HttpSession session=request.getSession();
        session.setAttribute("uname",n);
    out.print("<a href='servlet2'>visit</a>");
   out.close();
   }catch(Exception e){System.out.println(e);}
    }  }
```

SecondServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
  public class SecondServlet extends HttpServlet {
  public void doGet(HttpServletRequest request, HttpServletR
esponse response)
        try{
          response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        HttpSession session=request.getSession(false);
        String n=(String)session.getAttribute("uname");
        out.print("Hello "+n);
        out.close();
                }catch(Exception e){System.out.println(e);}
    }    }
```

Above example describes how to use the HttpSession object to find out the creation time and the last-accessed time for a session, number of times page is accessed. We would associate a new session with the request if one does not already exist.

```
protected void doGet(HttpServletRequest request, HttpServletResponse
response)throws ServletException, IOException {
HttpSession session = request.getSession(true);
Date createTime = new Date(session.getCreationTime());
Date lastAccessTime = new Date(session.getLastAccessedTime());
String title = "Welcome Back to my website";
 int visitCount = 0;      String visitCountKey = "visitCount";      String userIDKey =
"userID";      String userID = "ABCD";
if (session.isNew()) {
        title = "Welcome to my website";
        session.setAttribute(userIDKey, userID);
     } else {
        visitCount = (int)session.getAttribute(visitCountKey);
        visitCount = visitCount + 1;
        userID = (String)session.getAttribute(userIDKey);
     }
```

```
session.setAttribute(visitCountKey,  visitCount);
response.setContentType("text/html");
PrintWriter out = response.getWriter();
     out.println(title);
     out.println(session.getId());
     out.println(createTime);
     out.println(lastAccessTime);
     out.println(userID);
     out.println(visitCount);
}
```

Working with Files

- A Servlet can be used with an HTML form tag to allow users to upload files to the server. An uploaded file could be a text file or image file or any document.
- Creating a File Upload Form
- The following HTM code below creates an up loader form. Following are the important points to be noted down −
- The form **method** attribute should be set to **POST** method and GET method can not be used
- The form **enctype** attribute should be set to **multipart/form-data**.
- The form **action** attribute should be set to a servlet file which would handle file uploading at backend server. Following example is using **UploadServlet** servlet to upload file.
- To upload a single file you should use a single <input .../> tag with attribute type="file". To allow multiple files uploading, include more than one input tags with different values for the name attribute. The browser associates a Browse button with each of them.

Working with Files

```
<html> <head> <title>File Uploading Form</title> </head>

 <body> <h3>File Upload:</h3>

Select a file to upload: <br />

 <form action = "UploadServlet" method = "post" enctype = "multipart/form-data">

<input type = "file" name = "file" size = "50" /> <br />

<input type = "submit" value = "Upload File" />

</form> </body> </html>
```

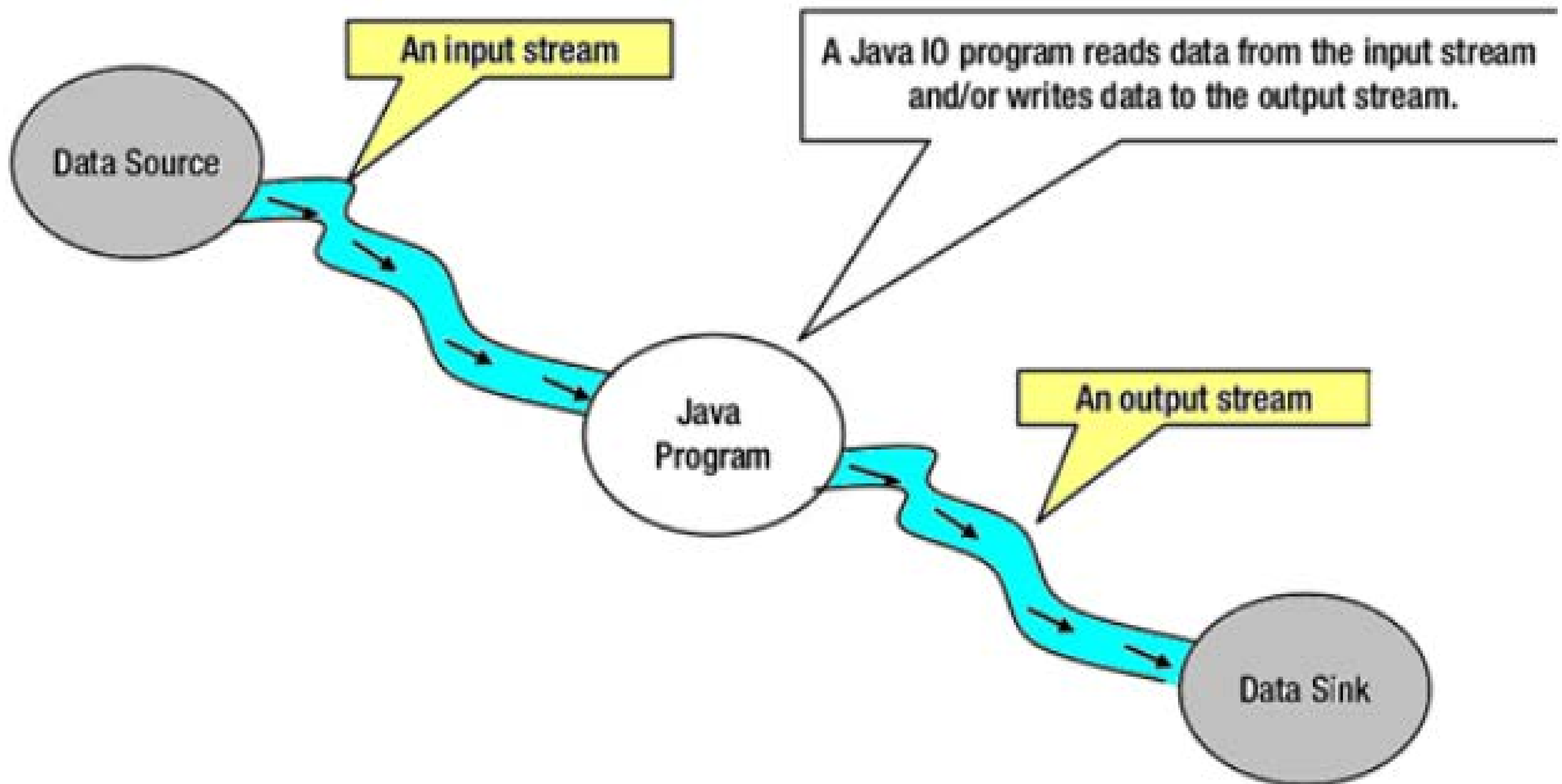
Working with Non-Blocking I/O

- **Blocking I/O**
- Blocking IO wait for the data to be write or read before returning. Java IO's various streams are blocking. It means when the thread invoke a write() or read(), then the thread is blocked until there is some data available for read, or the data is fully written. It is stream oriented.
- The I/O operation using this approach is slow.
- A stream can be used for **one-way** data transfer.

- **Non blocking I/O**
- Non blocking IO does not wait for the data to be read or write before returning. Java NIO non- blocking mode allows the thread to request writing data to a channel, but not wait for it to be fully written. The thread is allowed to go on and do something else in a mean time. It is buffer oriented.
- Data is read into a buffer from which it is further processed using a channel. In NIO we deal with the channel and buffer for I/O operation.
- A channel provides a **two-way** data transfer facility.

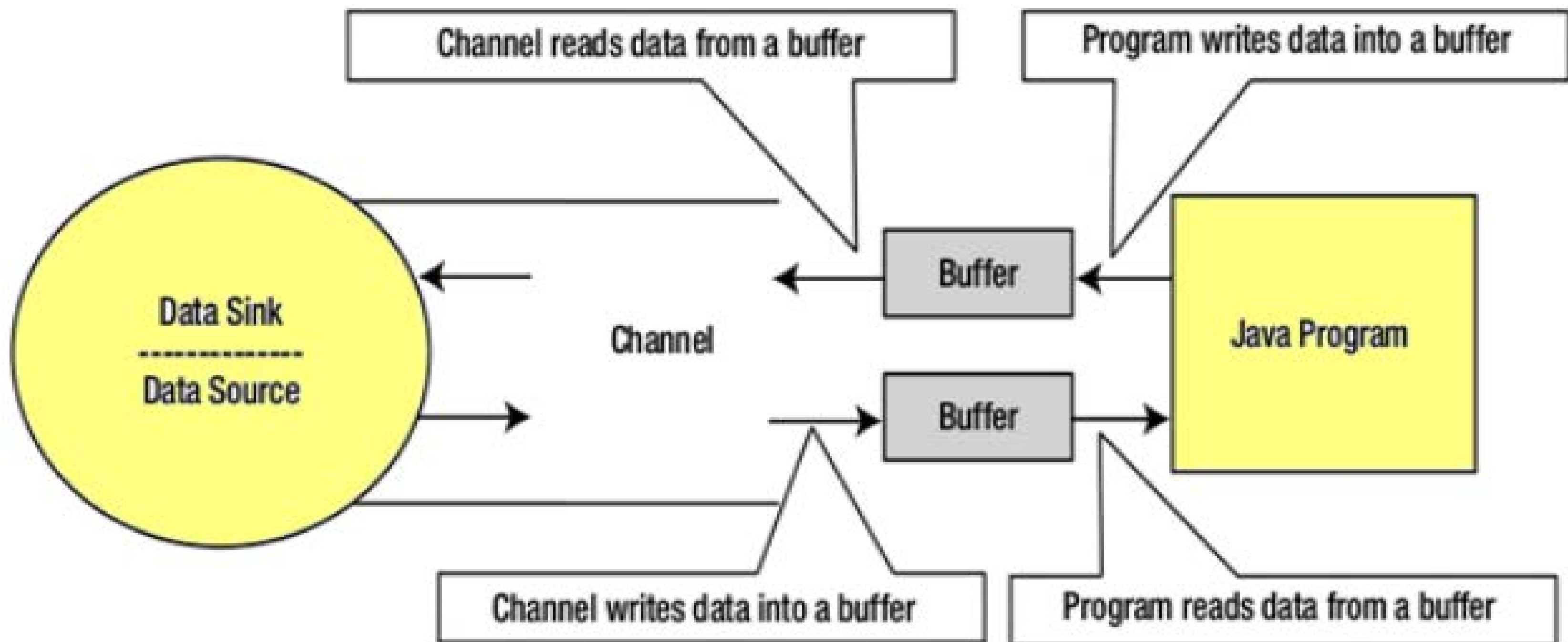
Stream Oriented

Java IO is stream oriented I/O means we need to read one or more bytes at a time from a stream. It uses streams for transferring the data between a data source/sink and a java program.



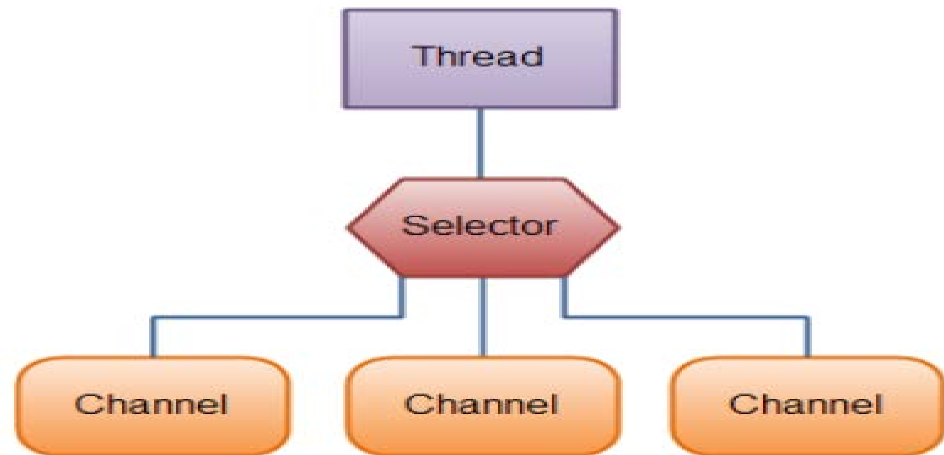
Buffer Oriented

Java NIO is buffer oriented I/O approach. Data is read into a buffer from which it is further processed using a channel. In NIO we deal with the channel and buffer for I/O operation.
The major difference between a channel and a stream is:
A stream can be used for **one-way** data transfer.
A channel provides a **two-way** data transfer facility.
Therefore with the introduction of channel in java NIO, the non-blocking I/O operation can be performed.



Buffer Oriented

Java NIO is buffer oriented I/O approach. Data is read into a buffer from which it is further processed using a channel. In NIO we deal with the channel and buffer for I/O operation.
The major difference between a channel and a stream is:
A stream can be used for **one-way** data transfer.
A channel provides a **two-way** data transfer facility.
Therefore with the introduction of channel in java NIO, the non-blocking I/O operation can be performed.

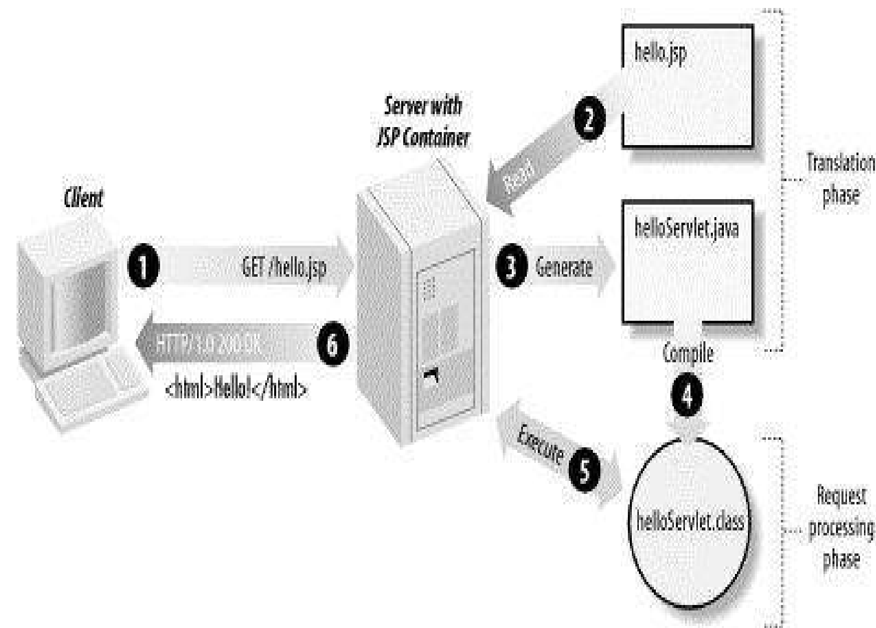


JSP

→ Mrs. Soniya Sharma.

JSP

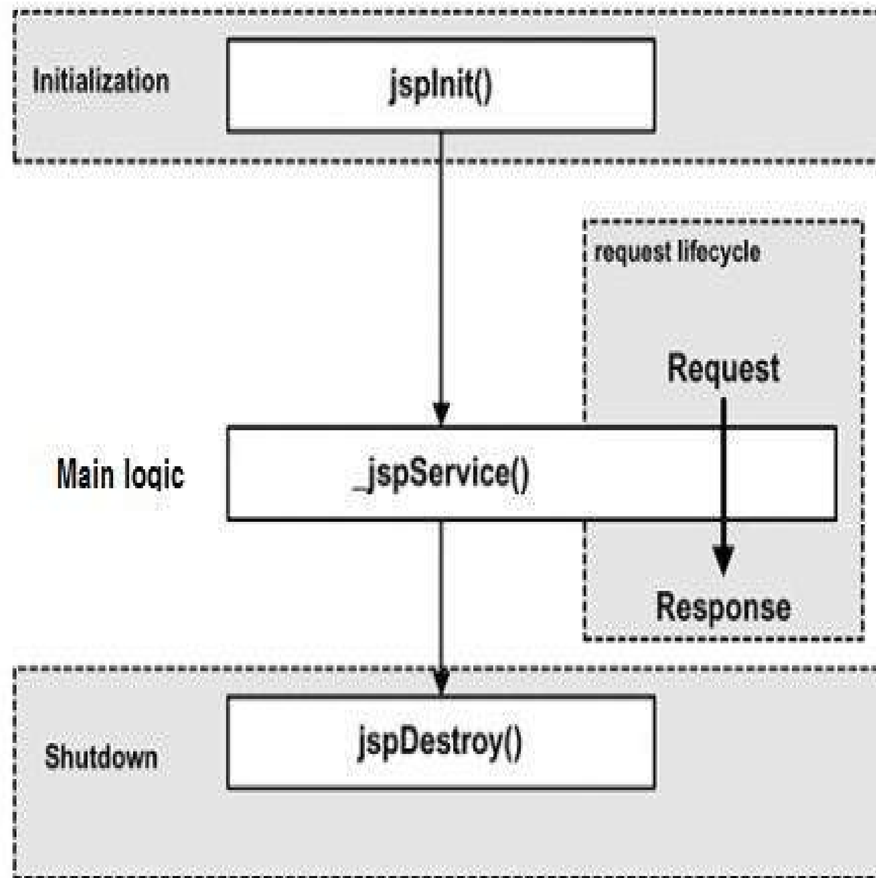
- JavaServer Pages (JSP) is a technology for developing Webpages that supports dynamic content. This helps developers insert java code in HTML pages by making use of special JSP tags.
- JSP Processing



JSP Life Cycle

- A JSP life cycle is defined as the process from its creation till the destruction. This is similar to a servlet life cycle with an additional step which is required to compile a JSP into servlet.
- The JSP life cycle includes following paths –

1. Compilation
2. Initialization
3. Execution
4. Cleanup


Initialization
jspInit()
request lifecycle
Request
Main logic
_jspService()
Response
Shutdown
jspDestroy()

JSP LifeCycle

1. Compilation :

- If the page has never been compiled, or if the JSP has been modified since it was last compiled, the JSP engine compiles the page.
- The compilation process involves three steps –
  - Parsing the JSP.
  - Turning the JSP into a servlet.
  - Compiling the servlet.

2. Initialization :

- When a container loads a JSP it invokes the **jspInit()** method before servicing any requests. If you need to perform JSP-specific initialization, override the **jspInit().**
- Initialization is performed only once.
- Generally initialize database connections, open files, and create lookup tables done in the jspInit().

JSP LifeCycle

3. <u>Execution</u> :

- Whenever a browser requests a JSP and the page has been loaded and initialized, the JSP engine invokes the **_jspService()**method in the JSP.
- The _jspService() method takes an **HttpServletRequest** and an **HttpServletResponse** as its parameters.
- The **_jspService()** method of a JSP is invoked on request basis.
- jspService() generats responses to all seven of the HTTP methods, i.e, **GET, POST, DELETE**, etc.

4. <u>Cleanup</u> :

- when you need to perform any cleanup, such as releasing database connections or closing open files, jspDestroy() is called which is equivalent to destroy() method of servlet.

JSP Pros & Cons

- Advantages of JSP

1. HTML friendly simple and easy language and tags.
2. Supports Java Code.
3. Supports standard Web site development tools.

- Disadvantages of JSP

1. As JSP pages are translated to servlets and compiled, it is difficult to trace errors occurred in JSP pages.

2. JSP pages require double the disk space to hold the JSP page because JSP pages are translated into class files, the server has to store the resultant class files with the JSP pages.

3. JSP pages must be compiled on the server when first accessed. This initial compilation produces a noticeable delay when accessing the JSP page for the first time.

Elements of JSP

1. The Scriptlet :

A scriptlet can contain any number of JAVA language statements, variable or method declarations, or expressions.

```
<% code %>
```

2. JSP Declarations :

variable or method must be declared before you use.

```
<%! int i = 0; %>
<%! int a, b, c; %>
<%! Circle a = new Circle(2.0); %>
```

Elements of JSP

3. JSP Expression:

```
<%= expression %>
<%= (new java.util.Date()).toLocaleString()%>
```

4. JSP Comments

```
<%-- This is JSP comment --%>
<!– HTML comment -->
```

Elements of JSP

5. JSP Directives

- A JSP directive affects the overall structure of the servlet class.
- There are three types of directive tag –

1. <%@ page ... %>

- Defines page-dependent attributes, such as scripting language, error page, and buffering requirements.

2. <%@ include ... %>

- Includes a file during the translation phase.

3. <%@ taglib ... %>

- Declares a tag library, containing custom actions, used in the page

Elements of JSP

6. **JSP Actions**

You can dynamically insert a file, reuse JavaBeans components, forward the user to another page, or generate HTML for the Java plugin.

```
<jsp:action_name attribute="value" />
```

a. **jsp:include -** Includes a file at the time the page is requested.
b. **jsp:useBean -** Finds or instantiates a JavaBean.
c. **jsp:setProperty -** Sets the property of a JavaBean.
d. **jsp:getProperty -** Inserts the property of a JavaBean into the output.
e. **jsp:forward -** Forwards the requester to a new page.
f. **jsp:text -** Used to write template text in JSP pages and documents.
g. **jsp:plugin -** Generates browser-specific code that makes an OBJECT or EMBED tag for the Java plugin.
h. **jsp:element -** Defines XML elements dynamically.
i. **jsp:attribute -** Defines dynamically-defined XML element's attribute.
j. **jsp:body -** Defines dynamically-defined XML element's body.

Elements of JSP

7. **JSP Implicit Objects** - JSP supports nine automatically defined variables, which are also called implicit objects.

i. r**equest -** This is the **HttpServletRequest** object associated with the request.

ii. response - This is the **HttpServletResponse** object associated with the response to the client.

iii. out - This is the **PrintWriter** object used to send output to the client.

iv. session - This is the **HttpSession** object associated with the request.

v. application - This is the **ServletContext** object associated with the application context.

vi. config - This is the **ServletConfig** object associated with the page.

vii. pageContext - This encapsulates use of server-specific features like higher performance **JspWriters**.

viii. page - This is simply a synonym for **this**, and is used to call the methods defined by the translated servlet class.

ix. Exception - The **Exception** object allows the exception data to be accessed by designated JSP.

Introduction to EJB

- EJB is an acronym for *enterprise java bean*. It is a specification provided by Sun Microsystems to develop secured, robust and scalable distributed applications.
- To run EJB application, you need an *application server* (EJB Container) such as Jboss, Glassfish, Weblogic, Websphere etc.
- It performs:
  - life cycle management,
  - security,
  - transaction management, and
  - object pooling.
- EJB application is deployed on the server, so it is called server side component also.

When use Enterprise Java Bean?

- **Application needs Remote Access**. In other words, it is distributed.
- **Application needs to be scalable**. EJB applications supports load balancing, clustering and fail-over.
- **Application needs encapsulated business logic**. EJB application is separated from presentation and persistent layer.

Types of Enterprise Java Bean

- There are 3 types of enterprise bean in java.
- Session Bean
  - Session bean contains business logic that can be invoked by local, remote or webservice client.
- Message Driven Bean
  - Like Session Bean, it contains the business logic but it is invoked by passing message.
- Entity Bean
  - It encapsulates the state that can be persisted in the database. It is deprecated. Now, it is replaced with JPA (Java Persistent API).

Uses of Message Driven Bean

- A single message-driven bean can process messages from multiple clients.
- They execute upon receipt of a single client message.
- Used in asynchronous communication.
- 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 used in stateless communication.

Interceptor

- Interceptors are perfect call-back methods in a situation where, we may want a certain method to be called after the bean has been instantiated but before it has been transferred to the client.
- Interceptor methods can be applied or bound at three levels.
- **Default** – Default interceptor is invoked for every bean within deployment.
- **Class** – Class level interceptor is invoked for every method of the bean.
- **Method**– Method level interceptor is invoked for a particular method of the bean.
- The preferred way to define interceptor in Java code is by using meta-data annotations.
- Some of the meta-data annotations found in the *javax.interceptor* package are: *@AroundInvoke*, *@AroundTimeout*, *@PostConstruct*, and *@PreDestroy*.

- *@AroundInvoke* is used to log a message whenever any method within the bean is entered
- *@PostConstruct* and *@PreDestroy* annotations are called life cycle interceptors in the sense that we can inform the container to invoke a method at a specific life cycle phase.
- Suppose we want to log an entry each time a bean is instantiated; we can use *@PostConstruct*. In a similar manner, if we want a log entry before a bean object is destroyed, we can use *@PreDestroy*.
- *@AroundTimeout it* Designates the method as a timeout interceptor, for interposing on timeout methods for enterprise bean timers.

Life Cycle of Interceptor

- Interceptor classes have the same lifecycle as their associated target class.
- When a target class instance is created, an interceptor class instance is also created for each declared interceptor class in the target class.
- That is, if the target class declares multiple interceptor classes, an instance of each class is created when the target class instance is created.
- The target class instance and all interceptor class instances are fully instantiated before any @PostConstruct callbacks are invoked, and any @PreDestroy callbacks are invoked before the target class and interceptor class instances are destroyed.

JNDI

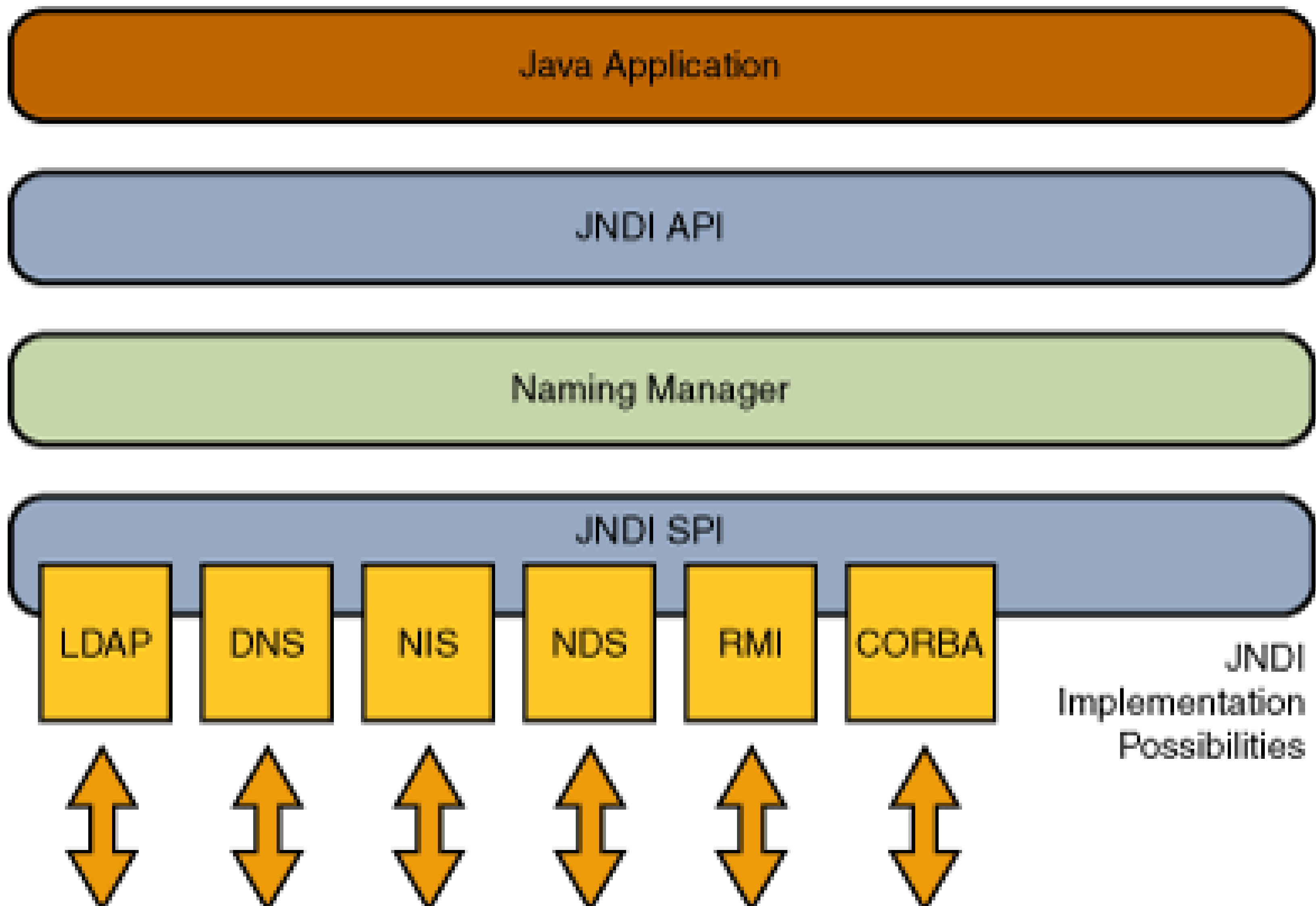
- Finding resources is of particular importance in large-scale enterprise environments, where the applications you build may depend on services provided by applications written by other groups in other departments.
- The Java Naming and Directory Interface (JNDI). JNDI provides a common interface to many existing naming services.
- The **Java Naming and Directory Interface** (**JNDI**) is a Java API for a directory service that allows Java software clients to discover and look up data and resources (in the form of Java objects) via a name.
- JNDI is independent of the underlying implementation.

JNDI

- The API provides:
  - a mechanism to bind an object to a name.
  - a directory-lookup interface that allows general queries.
  - an event interface that allows clients to determine when directory entries have been modified.
  - LDAP extensions to support the additional capabilities of an LDAP service
- Typical uses of JNDI include:
  - connecting a Java application to an external directory service (such as an address database or an LDAP (lightweight directory access protocol)server)
  - allowing a Java Servlet to look up configuration information provided by the hosting web container[2]

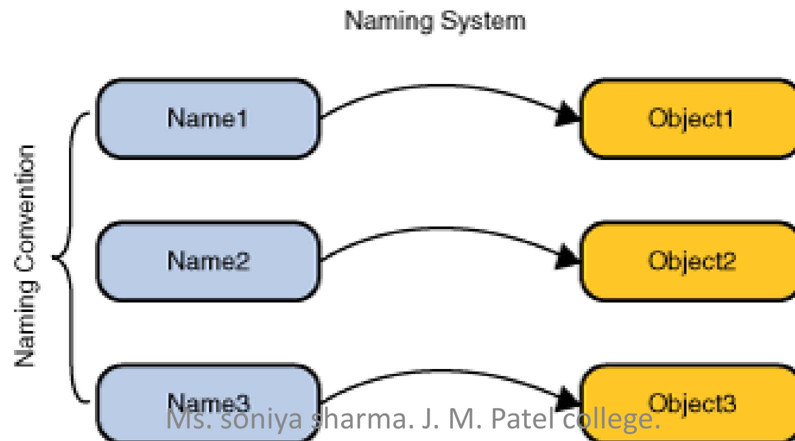
JNDI Architecture

- JNDI has an API and an SPI(Service Provider Interface). The API enables the Java applications to access the naming and directory services. The SPI enables to plug in various naming and directory services. The Java applications can use these services using the API. This concept is explained below as a schematic.
- The service providers **LDAP**(**Lightweight Directory Access Protocol**),**RMI**(**Remote Method Invocation**) **Registry** , **CORBA** (**Common Object Request Broker Architecture**)are inbuilt with Java from SDK v1.3 onwards.These are inbuilt by default.


Java Application
JNDI API
Naming Manager
JNDI SPI
LDAP
DNS
NIS
NDS
RMI
CORBA
JNDI
Implementation
Possibilities

Naming Service

- A naming service maps developer friendly names to objects . So that the applications can access the same object with the bound name.
- ***Names*** –To do the look up , one must know the name of object. The syntax of naming is determined by the naming service.It is known as the ***naming convention .***
- ***Bindings –*** The association between a name and an object is referred as *binding* .If an application need to access an object using a name then the object should bound to the name before.
- ***Context*** – A context is a set of name to object bindings. The context provides the lookup. It also provides options to bind ,unbind and rebind objects .
- **InitilalContext** –It is the starting point of all naming and directory operations.
- **Exceptions** : JNDI defines a hierarchy of Exceptions . The super class is **NamingExceptions**. All other exceptions are deriving from **NamingException.**
- The ***javax.naming*** package contains the classes and interfaces for accessing naming services.

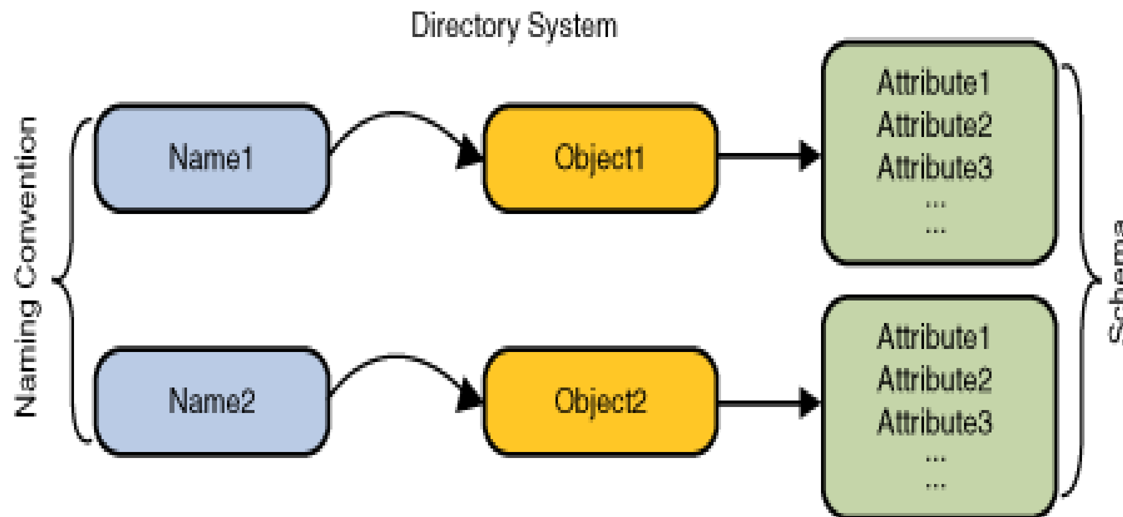


Directory Service

- Similar to naming service here also it is possible to access objects with its name. In addition to this , it is possible to get the attributes of objects. Also it is possible to search for objects with attributes.
- **Directory and Directory Service**
- A ***directory*** is a connected set of directory objects.A ***directory service*** is a service which provides options for creating , adding ,removing and modifying attributes associated with directory.
- To use JNDI concepts in our application , we should have the naming classes along with at least one service providers. Java provides inbuilt support for:
  - ***Lightweight Directory Access Protocol (LDAP)***
  - ***Common Object Request Broker Architecture (CORBA) Common Object Services (COS) name service***
  - ***Java Remote Method Invocation (RMI) Registry***
- If we need to use other services in our application , then we need to use the JNDI implementation for the same.

Directory Service

- A directory service associates names with objects and also associates such objects with *attributes*.
- directory service = naming service + objects containing attributes
- You not only can look up an object by its name but also get the object's attributes or *search* for the object based on its attributes.

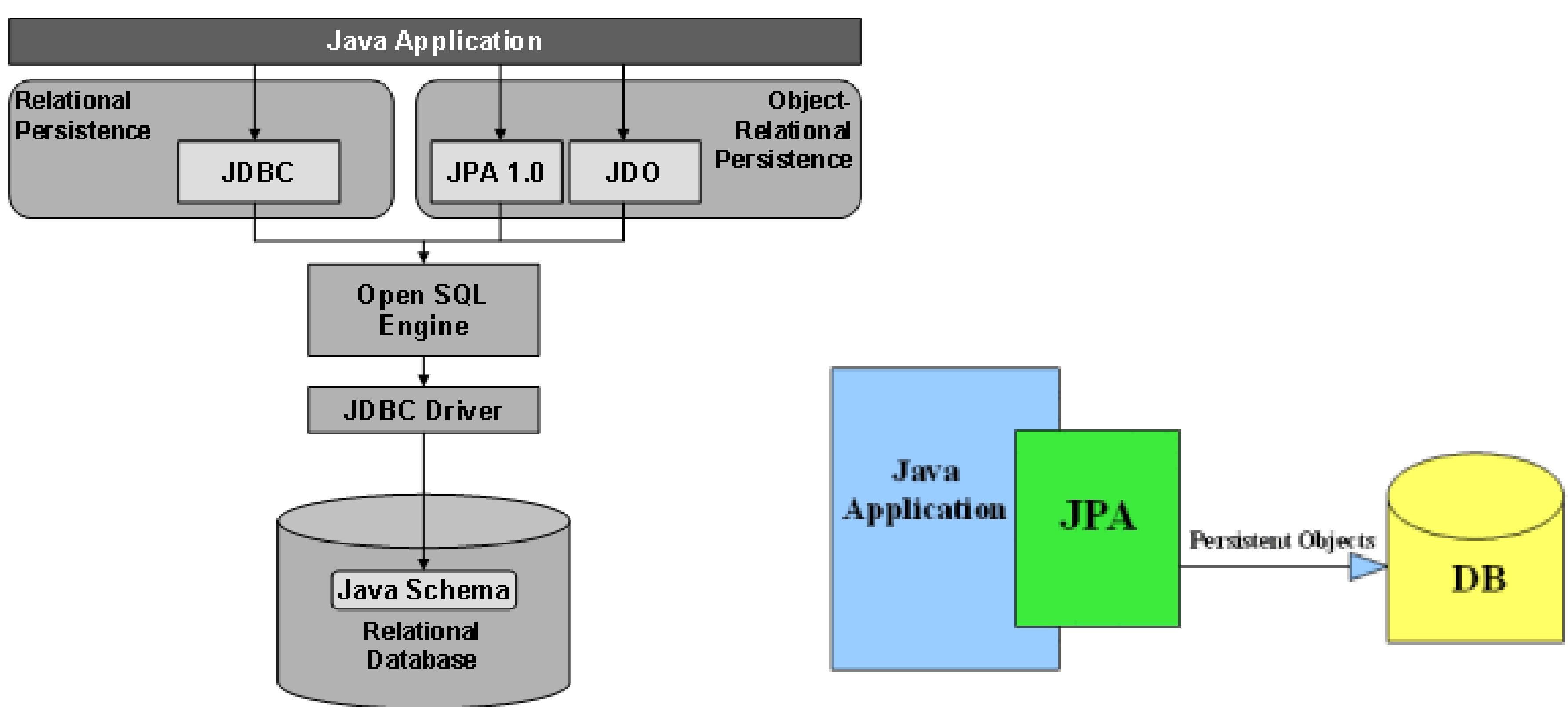


Persistence in JAVA

- Data Persistence is a means for an application to persist and retrieve information from a non-volatile storage system.
- Persistence is vital to enterprise applications because of the required access to relational databases.
- Applications that are developed for this environment must manage persistence themselves or use third-party solutions to handle database updates and retrievals with persistence.
- Java persistence could be defined as storing anything to any level of persistence using the Java programming language.
- There are many ways to make data persist in Java, including (to name a few): JDBC, serialization, file IO, object databases, and XML databases.
- Java Persistence API (JPA) provides a mechanism for managing persistence and object-relational mapping and functions since the EJB 3.0 specifications.
- However, the majority of data is persisted in databases, specifically relational databases. Most things that you do on a computer or web site that involve storing data, involve accessing a relational database. Relational databases are the standard mode of persistent storage for most industries, from banking to manufacturing.

Persistence in JAVA

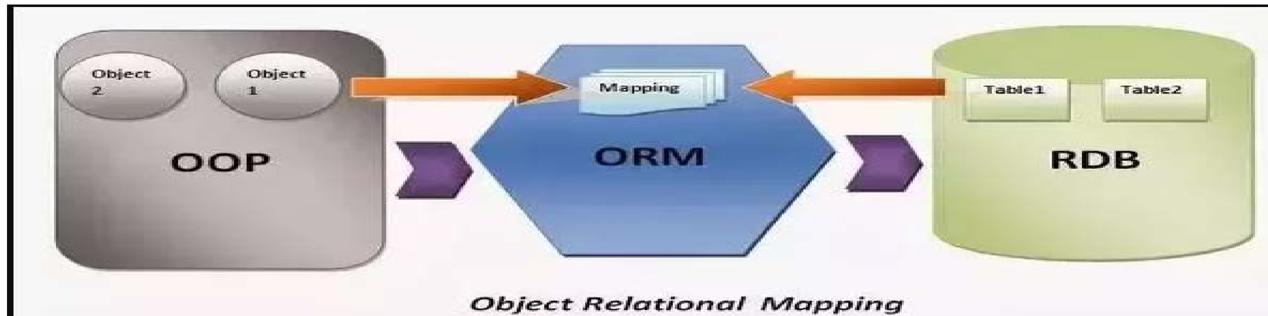
- There are many things that can be stored in databases with Java. Java data includes strings, numbers, date and byte arrays, images, XML, and Java objects.
- Many Java applications use Java objects to model their application data. Because Java is an Object Oriented language, storing Java objects is a natural and common approach to persisting data from Java.

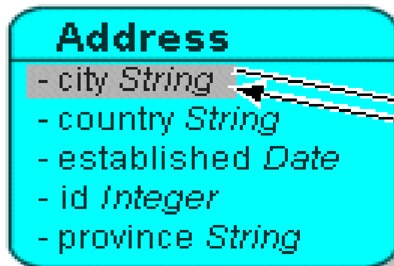


Current Persistence standards in java

- Data Persistence is a means for an application to persist and retrieve information from a non-volatile storage system.
- There are many ways to make data persist in Java as follows:
- Serialization :
  - Serialization is the conversion of object to a series of bytes, so that the object can be easily saved to persistent storage or streamed across a communication link. The byte stream can then be de-serialized and converted into a replica of original object.
  - Limitation – it can not be used with large size objects.555
- JDBC :
  - It gives full access to SQL database functionality. JDBC requires the developer to explicitly manage the values of fields and to map them into relational database tables.
  - Limitation – the interaction of JDBC to database is totally dependent on SQL execution.
- EJB Entity Beans :
  - An entity bean represents the persistent data stored in the database. It is server side component. An entity bean can manage its own persistence (Bean managed persistence) or can delegate this function to its EJB Container (Container managed persistence).
  - Limitation – require one to one mapping, require huge application server to run , need developer to map field and table column.
- JPA (Java Persistence API) :
  - It is a Java application programming interface specification that describes the management of relational data in applications using Java Platform, Standard Edition and Java Platform, Enterprise Edition.

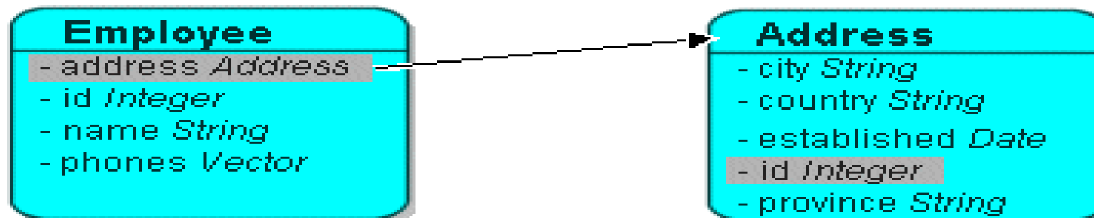
Object Relational Mapping





EMP_ID	CITY	COUNTRY	EST_DATE	PROVINCE
274	Ottawa	Canada	01/01/1995	Ottawa
105	Toronto	Canada	04/01/1993	Tornoto
421	New York	USA	08/01/2001	New York

Java Class Relational Database



Java Class (one to one relationship)

EMPLOYEE table

EMP_ID	NAME	ADDR_ID
103	John Doe	305
104	Jane Smith	226
105	Tom Jones	274

ADDRESS table

ADD_ID	CITY	COUNTRY	EST_DATE	PROVINCE
105	Ottawa	Canada	01/01/1995	Ottawa
274	Toronto	Canada	04/01/1993	Toronto
421	New York	USA	08/01/2001	New York

Ms. soniya sharma, J. M. Patel College

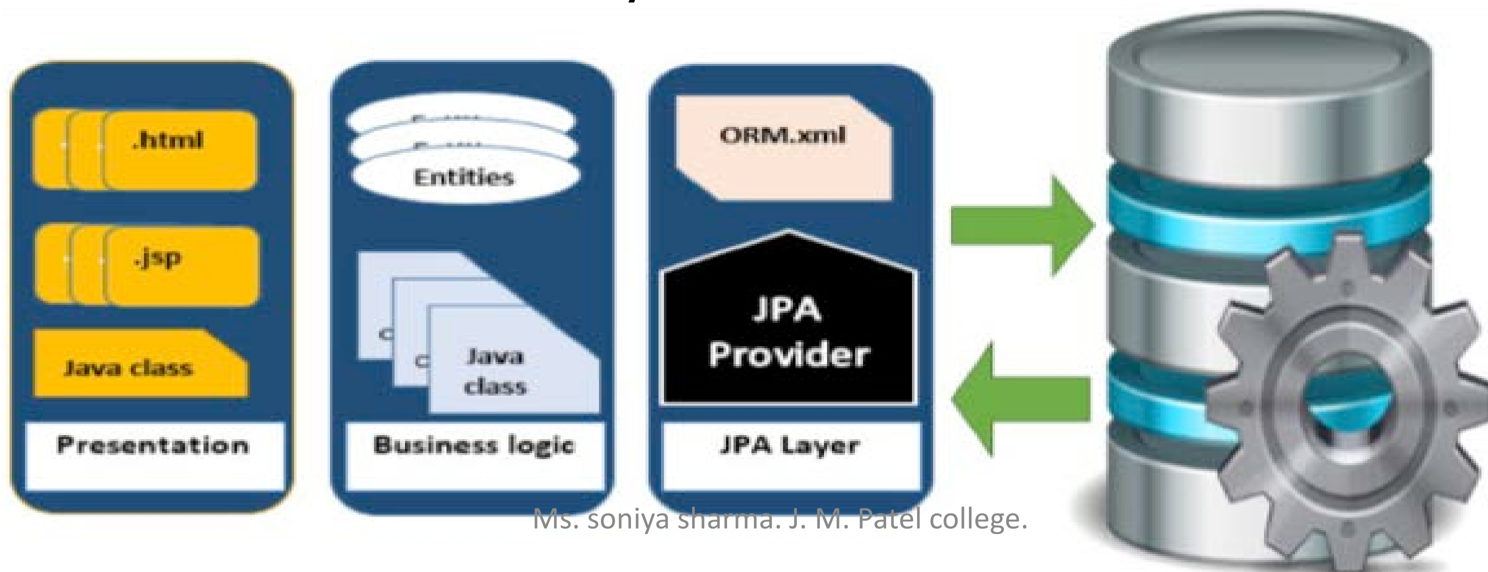
Relational Database (one to one relationship)

ORM (Object Relation Mapping)

- When we work with an object-oriented system, there is a mismatch between the object model and the relational database. RDBMSs represent data in a tabular format whereas object-oriented languages, such as Java or C# represent it as an interconnected graph of objects.
- ORM stands for **O**bject-**R**elational **M**apping (ORM) is a programming technique for converting data between relational databases and object oriented programming languages such as Java, C#, etc.
- To make programming easier and reduce the chance of making mistakes, many developers prefer not to execute SQL statements directly, but to build an object model that reflects the data structure. In runtime, data will be retrieved from database and filled into the object model. Developers can then work entirely with objects, without writing any SQL statements.
- The technique to convert data between object model and relational database is known as object-relational mapping (ORM, O/RM and O/R mapping).

Java Persistence API (JPA)

- Java Persistence API is a collection of classes and methods to persistently store the vast amounts of data into a database which is provided by the Oracle Corporation.
- To reduce the burden of writing codes for relational object management, a programmer follows the 'JPA Provider' framework, which allows easy interaction with database instance. Here the required framework is taken over by JPA.
- JPA acts as a bridge between object-oriented domain models and relational database systems.



JPA Versions / Specification

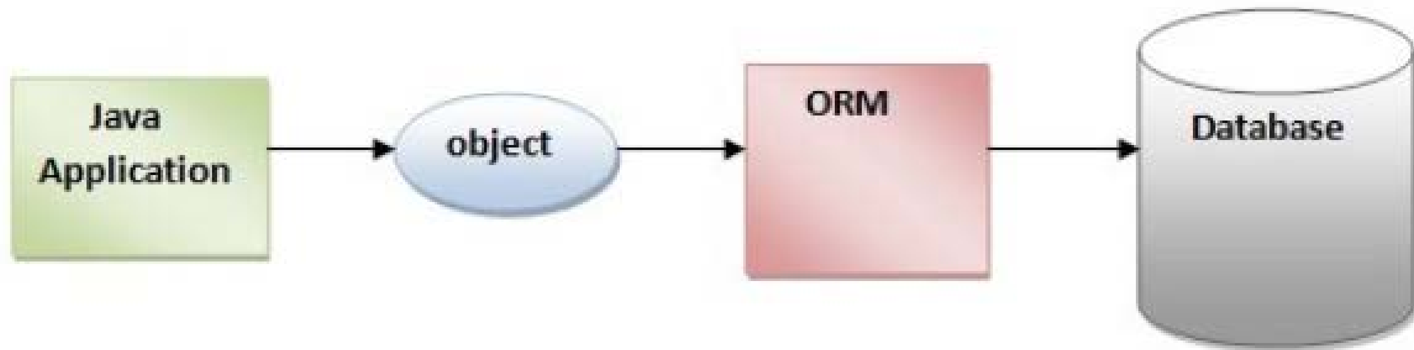
- The first version of Java Persistence API, JPA 1.0 was released in 2006 as a part of EJB 3.0 specification.
- Following are the other development versions released under JPA specification: -
- JPA 2.0 - This version was released in the last of 2009. Features : -
  - It supports validation.
  - It expands the functionality of object-relational mapping.
  - It shares the object of cache support.
- JPA 2.1 - The JPA 2.1 was released in 2013 with the following features:
  - It allows fetching of objects.
  - It provides support for criteria update/delete.
- JPA 2.2 - The JPA 2.2 was released as a development of maintenance in 2017. Some of its important feature are: -
  - It supports Java 8 Date and Time.
  - It provides @Repeatable annotation that can be used when we want to apply the same annotations to a declaration or type use.

Software Requirements

- From application development perspective, the following software will be required on the development machine:
  - Java Development Kit
  - Net Beans IDE
  - My SQL Community Server
  - JDBC driver for MYSQL
  - Sun Glassfish Enterprise Server
  - Library Files: The Java Library [.JAR] i.e. JDBC driver is required. This will be specific to relational database to be used.

Introduction to Hibernate

- 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.
- **ORM Tool-** 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.



- **JPA -** Java Persistence API (JPA) is a Java specification that provides functionality and standard to ORM tools. The **javax.persistence** package contains JPA classes and interfaces.

Advantages of Hibernate Framework

1) **Open Source and Lightweight**

- Hibernate framework is open source and lightweight.

2) **Fast Performance**

- The performance of hibernate framework is fast because cache is internally used in hibernate framework.

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.

Introduction to Hibernate

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

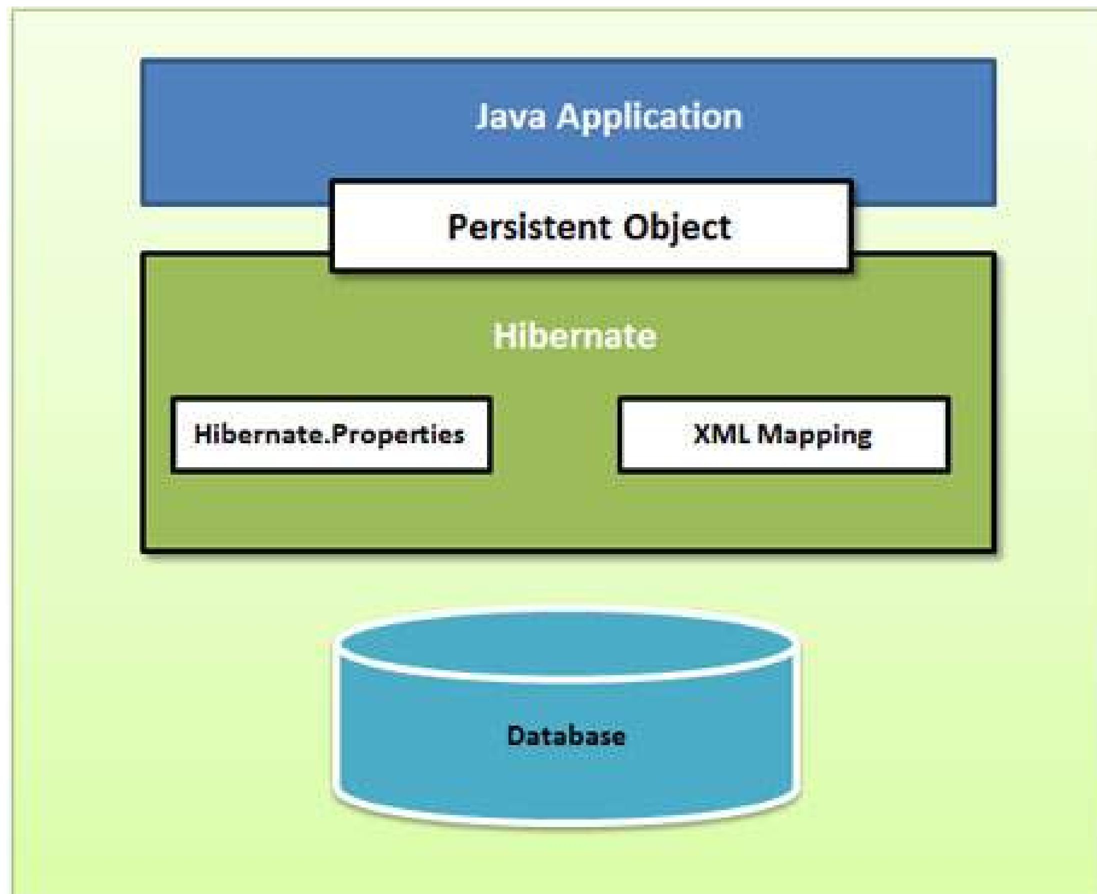
- Fetching data from multiple tables is easy in hibernate framework.

6) Provides Query Statistics and Database Status

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

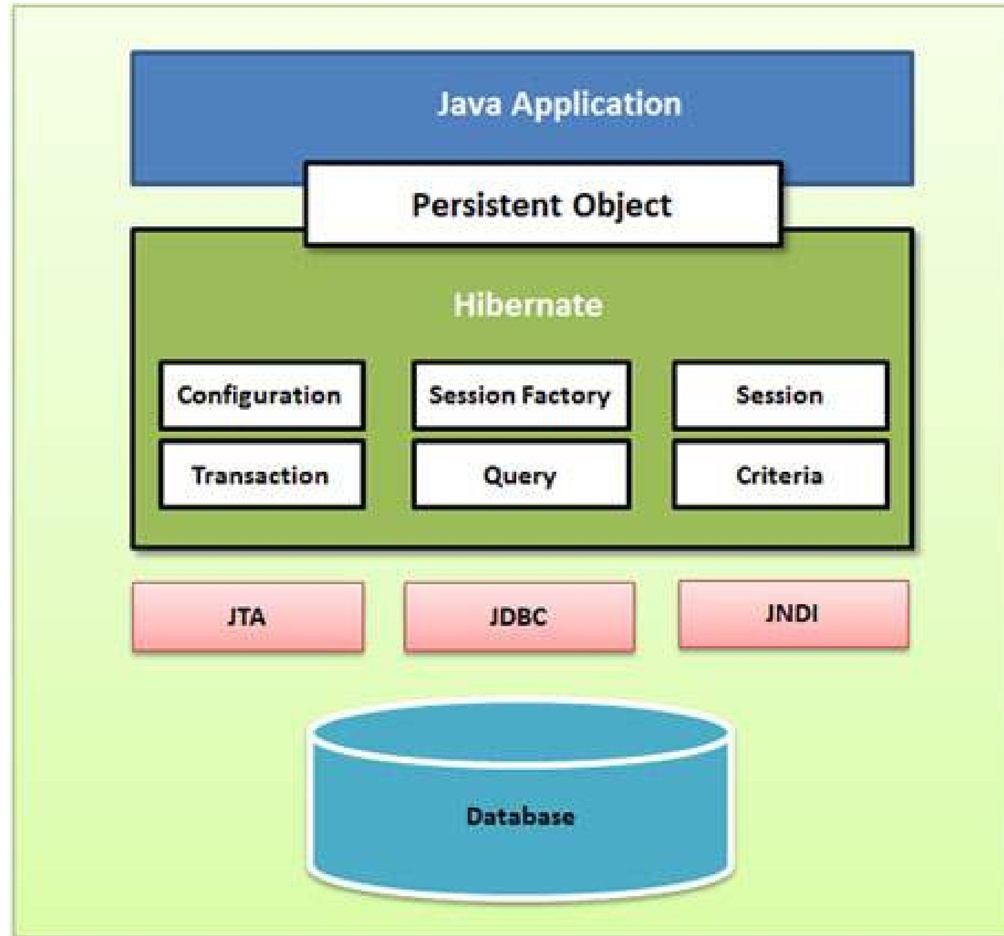
Hibernate Architecture

- Hibernate has a layered architecture which helps the user to operate without 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.



Hibernate Architecture

- Following is a detailed view of the Hibernate Application Architecture with its important core classes.



Hibernate Architecture

- Hibernate uses various existing Java APIs, like JDBC, Java Transaction API(JTA), and Java Naming and Directory Interface (JNDI).
- **Configuration Object**
- The Configuration object is the first Hibernate object you create in any Hibernate application. It is 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 in turn configures Hibernate 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.

Hibernate Architecture

- The SessionFactory is a heavyweight object. 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.
- **Transaction Object**
- A Transaction represents a unit of work with the database and most of the RDBMS supports transaction functionality.
- **Query Object**
- Query objects use SQL or Hibernate Query Language (HQL) string to retrieve data from the database and create objects.
- **Criteria Object**
- Criteria objects are used to create and execute object oriented criteria queries to retrieve objects.

Hibernate Architecture

- The SessionFactory is a heavyweight object. 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.
- **Transaction Object**
- A Transaction represents a unit of work with the database and most of the RDBMS supports transaction functionality.
- **Query Object**
- Query objects use SQL or Hibernate Query Language (HQL) string to retrieve data from the database and create objects.
- **Criteria Object**
- Criteria objects are used to create and execute object oriented criteria queries to retrieve objects.

How Hibernate Works?