Course Code	**:**	**MCS-051**
Course Title	**:**	**Advanced Internet technologies**
Assignment Number	**:**	**MCA(V)-051/Assignment/2018-19**
Maximum Marks	**:**	**100**
Weightage	**:**	**25%**

Last Dates for Submission :15th October, 2018 (For July session)
15th April, 2019 (For January session)

Question1: Make a JSP page that randomly select a background color for each request.

Ans.

```
<%

String bgColor = "";

if ( Math.random() < 0.3 ) {

bgColor = "GREEN";

}

else if ( (Math.random() > 0.3) && (Math.random() < 0.5) ){

bgColor = "BLUE";

}

else if ( (Math.random() > 0.5) && (Math.random() < 0.8) ){

bgColor = "RED";

}

%>

<BODY BGCOLOR="<%= bgColor %>">
```

Question2: Explain the benefits offered by EJB component architecture to application developers and customers.

Ans.

Enterprise Java Beans (EJB):

EJB is written in Java programming language and it is server side component that encapsulates the business logic of an java application from customers. Business logic code is managed in EJB that fulfills the purpose of the java applications. For example, in an eCommerce web application, the enterprise java beans might implement the business logic in methods bgcalled checkAvailableItem and orderProduct.

Benefits of Enterprise Java Beans EJB

EJB technology enables rapid and simplified the process of distributed, transactional, secure and portable java desktop applications development and Java ee web applications development because

- EJB container provides System level services to enterprise java beans.
- EJB developer just focus on business logic and on solving business problems.
- Because business logic lies in EJB, so Front end developer can focus on the presentation of client interface.
- The client developer does not have to code the routines that implement business rules or access databases. As a result, clients side has less codes which is particularly important for clients that run on small devices.
- Java Beans are portable components which enable the java application assembler to build new applications from existing java beans.
- EJB is a standard API due to which applications build on EJB can run on any complaint Java EE web application server.

Types of Enterprise Java Beans:

There are two main types of EJBs

1- Session Beans

2- Message Driven Beans

1- Session Beans

A session bean encapsulates business logic that can be invoked programmatically by a client over local, remote, or web service client views. A session bean is not persistent, means its data is not saved to a database. EJB Session Beans has three types which are

Stateful Session Bean

In stateful session bean, 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.

Stateless Session Bean

In EJB stateless session bean conversational state is not maintained with client. When a client invokes the methods of a stateless bean, the bean's instance variables may contain a state specific to that client but only for the duration of the invocation.

Single tone Session Bean

A singleton session bean is instantiated once per application and exists for the whole lifecycle of the java application. A single enterprise bean instance is shared across all the applications clients and it is concurrently accessed by clients

2- Message Driven Bean (MDB)

Message Driven Beans (MDBs) also known as Message Beans. Message Driven Beans are business objects whose execution is triggered by messages instead of by method calls.

Question 3: What are the benefits of using entity bean over directly using JDBC APIs to do database operations? Also discuss when should we use one over the other.

Ans.

Entity Beans actually represents the data in a database. It is not that Entity Beans replaces JDBC API. There are two types of Entity Beans Container Managed and Bean Mananged.

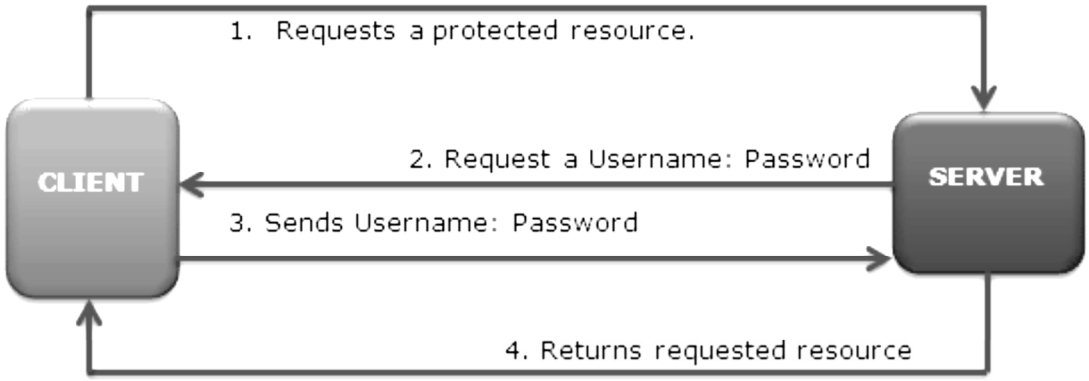
In Container Managed Entity Bean - Whenever the instance of the bean is created the container automatically retrieves the data from the DB/Persistance storage and assigns to the object variables in bean for user to manipulate or use them. For this the developer needs to map the fields in the database to the variables in deployment descriptor files (which varies for each vendor).

In the Bean Managed Entity Bean - The developer has to specifically make connection, retrive values, assign them to the objects in the ejbLoad() which will be called by the container when it instatiates a bean object. Similarly in the ejbStore() the container saves the object values back the the persistance storage. ejbLoad and ejbStore are callback methods and can be only invoked by the container.

Question4: Explain four basic mechanisms through which a web client can authenticate a user to a web server during HTTP authentication.

Ans.

The authentication mechanism is very useful when we try to access a protected web resource; at that time the web container activates the authentication mechanism that has been configured for that resource. A web client can authenticate a user to a web server using one of the following mechanisms:
HTTP Basic Authentication
HTTP Digest Authentication
Form Based Authentication
HTTPS Client Authentication
HTTP Basic Authentication: HTTP basic authentication is defined by the HTTP specification that lightly sends the user's user name and password over the Internet as text that is uu-encoded (Unix-to-Unix encoded) but not encrypted. If someone can intercept the transmission, the user name and password informationcan easily be decoded. It should only be used with HTTPS, as the password can be easily captured and reused over HTTP. Basic authentication is supported by Exchange 2000 Server and Exchange Server 2003.
With basic authentication, the following things occur:
1. A client requests access to a protected resource.
2. The web server returns a dialog box that requests the user name and password.
3. The client submits the user name and password to the server.
4. The server validates the credentials and, if successful, returns the requested resource.

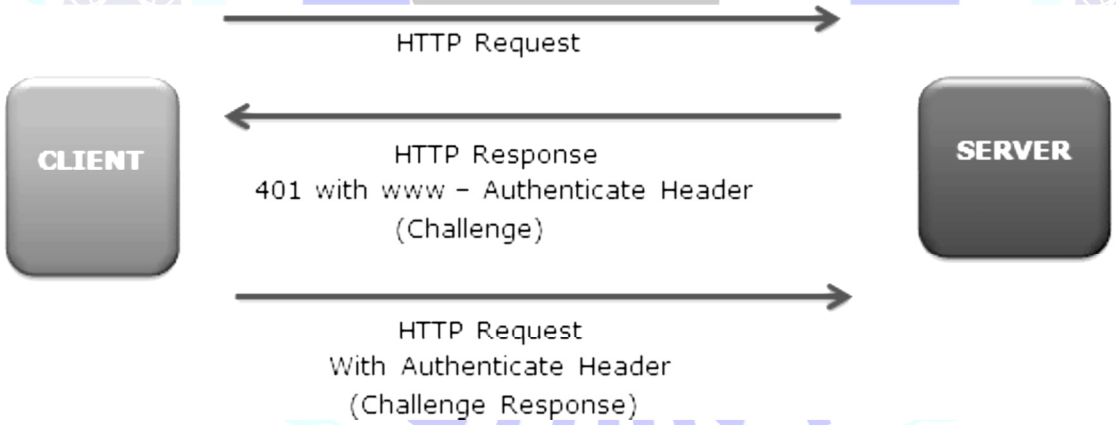


HTTP Basic Authentication

HTTP Digest Authentication:

Similar to HTTP Basic Authentication, HTTP Digest Authentication authenticates a user based on a username and a password. As Digest Authentication is not currently in widespread use, servlet containers are encouraged but NOT REQUIRED to support it. The advantage of this method is that the clear text password is protected in transmission; it cannot be determined from the digest that is submitted by the client to the server. Digested password authentication supports the concept of digesting user passwords. This causes the stored version of the passwords to be encoded in a form that is not easily reversible, but that the web server can still utilize for authentication.

The difference between basic and digest authentication is that on the network connection between the browser and the server, the passwords are encrypted, even on a non-SSL connection. Digested password is authentication based on the concept of a hash or digest. In this stored version, the passwords are encoded in a form that is not easily reversible and this is used for authentication.



Question5: What is DTD? Why do we use it? Write a XML DTD to represent for following product details :
product –ID
Type of products:- five different types of product
price,
discount offer –(Yes / No)

Ans.

A document type definition (DTD) is a set of markup declarations that define a document type for an SGML-family markup language (SGML, XML, and HTML).
A Document Type Definition (DTD) defines the legal building blocks of an XML document. It defines the document structure with a list of legal elements and attributes. A DTD can be declared inline inside an XML document, or as an external reference.
XML uses a subset of SGML DTD.

```
<?xml version = "1.0" standalone="yes"?>

<document>

<productinfo >

<product>Printer</product>

<id>111</id>

<price>10111.00</price>

</productinfo>

< productinfo >

<product>Laptop</product>

<id>222</id>

<price>99989.00</price>

</ productinfo >

< productinfo >

<product>Mobile </product>

<id>333</id>

<price>19989.00</price>

</ productinfo >

< productinfo >

<product>FAX Machine </product>

<id>444</id>

<price>9000.00</price>

</ productinfo >

< productinfo >

<product>Notebook</product>
```

```
<id>555</id>

<price>80009.00</price>

</ productinfo >

</document>
```

Question6: Write a web based feedback application where the registered customers should be able to login with the customer-ID and provide a feedback about the product. Design a suitable form and do coding of the buttons. You are required to use JSP, Servlet and JDBC.

Ans.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<!DOCTYPE html >
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8888-3">
<title> Registration Form</title>
</head>
<body>
<h1> Register Form</h1>
<form action="my_register" method="post">
<table style="with: 50%">
<tr>
<td>First Name</td>
<td><input type="text" name="first_name" /></td>
</tr>
<tr>
<td>Last Name</td>
<td><input type="text" name="last_name" /></td>
</tr>
<tr>
<td>UserName</td>
<td><input type="text" name="username" /></td>
</tr>
<tr>

<td>Password</td>

<td><input type="password" name="password" /></td>
</tr>
<tr>
<td>Address</td>
<td><input type="text" name="address" /></td>
</tr>
<tr>
```

```
<td>Contact No</td>
<td><input type="text" name="contact" /></td>
</tr></table>
<input type="submit" value="Submit" /></form>
</body>
</html>
```

my_register.java

```
package demotest;
import java.io.IOException;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
/**
*/

public class my_register extends HttpServlet {
private static final long serialVersionUID = 1L;
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
// TODO Auto-generated method stub
String first_name = request.getParameter("first_name");
String last_name = request.getParameter("last_name");
String username = request.getParameter("username");
String password = request.getParameter("password");
String address = request.getParameter("address");
String contact = request.getParameter("contact");
if(first_name.isEmpty() || last_name.isEmpty() || username.isEmpty() ||
password.isEmpty() || address.isEmpty() || contact.isEmpty())
{
RequestDispatcher req = request.getRequestDispatcher("register_1.jsp");
req.include(request, response);
}
else
{
RequestDispatcher req = request.getRequestDispatcher("register_2.jsp");
req.forward(request, response);
}
}

}
```

LOGIN FORM

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<!DOCTYPE html >
<html>
<head>
```

```
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8888-3">
<title> Login Form</title>
</head>
<body>
<form action="my_login" method="post">
<table style="with: 50%">
<tr>
<td>UserName</td>
<td><input type="text" name="username" /></td>
</tr>
<tr>
<td>Password</td>
<td><input type="password" name="password" /></td>
</tr>
</table>
<input type="submit" value="Login" /></form>

</body>
</html>
```

my_login.java(servlet)

```
package demotest;
import java.io.IOException;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
/**
* Servlet implementation class my_login
*/
public class my_login extends HttpServlet {
public my_login() {
super();
// TODO Auto-generated constructor stub
}
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {

// TODO Auto-generated method stub
String username = request.getParameter("username");
String password = request.getParameter("password");
if(username.isEmpty() || password.isEmpty() )
{
RequestDispatcher req = request.getRequestDispatcher("register_3.jsp");
req.include(request, response);
}
else
{
```

```
RequestDispatcher req = request.getRequestDispatcher("register_4.jsp");
req.forward(request, response);
}
}
}
```

Question7: What are the advantages of using Java's multilayer security implementation.

Ans.

Java EE Security Implementation Mechanisms

Java EE security services are provided by the component container and can be implemented using declarative or programmatic techniques (container security is discussed more in Securing Containers). Java EE security services provide a robust and easily configured security mechanism for authenticating users and authorizing access to application functions and associated data at many different layers. Java EE security services are separate from the security mechanisms of the operating system.

Application-Layer Security

In Java EE, component containers are responsible for providing application-layer security. Application-layer security provides security services for a specific application type tailored to the needs of the application. At the application layer, application firewalls can be employed to enhance application protection by protecting the communication stream and all associated application resources from attacks.

Java EE security is easy to implement and configure, and can offer fine-grained access control to application functions and data. However, as is inherent to security applied at the application layer, security properties are not transferable to applications running in other environments and only protect data while it is residing in the application environment. In the context of a traditional application, this is not necessarily a problem, but when applied to a web services application, where data often travels across several intermediaries, you would need to use the Java EE security mechanisms along with transport-layer security and message-layer security for a complete security solution.

The advantages of using application-layer security include the following:

- Security is uniquely suited to the needs of the application.
- Security is fine-grained, with application-specific settings.

The disadvantages of using application-layer security include the following:

- The application is dependent on security attributes that are not transferable between application types.
- Support for multiple protocols makes this type of security vulnerable.
- Data is close to or contained within the point of vulnerability.

Transport-Layer Security

Transport-layer security is provided by the transport mechanisms used to transmit information over the wire between clients and providers, thus transport-layer security relies on secure HTTP transport (HTTPS) using Secure Sockets Layer (SSL). Transport security is a point-to-point security mechanism that can be used for authentication, message integrity, and confidentiality. When running over an SSL-protected session, the server and client can authenticate one another and negotiate an

encryption algorithm and cryptographic keys before the application protocol transmits or receives its first byte of data. Security is "live" from the time it leaves the consumer until it arrives at the provider, or vice versa, even across intermediaries. The problem is that it is not protected once it gets to its destination. One solution is to encrypt the message before sending.
Transport-layer security is performed in a series of phases, which are listed here:
 The client and server agree on an appropriate algorithm.
 A key is exchanged using public-key encryption and certificate-based authentication.
 A symmetric cipher is used during the information exchange.

The advantages of using transport-layer security include the following:

Relatively simple, well understood, standard technology.
 Applies to message body and attachments.
The disadvantages of using transport-layer security include the following:
 Tightly-coupled with transport-layer protocol.
 All or nothing approach to security. This implies that the security mechanism is unaware of message contents, and as such, you cannot selectively apply security to portions of the message as you can with message-layer security.
 Protection is transient. The message is only protected while in transit. Protection is removed automatically by the endpoint when it receives the message.
 Not an end-to-end solution, simply point-to-point.

Message-Layer Security
In message-layer security, security information is contained within the SOAP message and/or SOAP message attachment, which allows security information to travel along with the message or attachment. For example, a portion of the message may be signed by a sender and encrypted for a particular receiver. When the message is sent from the initial sender, it may pass through intermediate nodes before reaching its intended receiver. In this scenario, the encrypted portions continue to be opaque to any intermediate nodes and can only be decrypted by the intended receiver. For this reason, message-layer security is also sometimes referred to as **end-to-end security**.
The advantages of message-layer security include the following:
 Security stays with the message over all hops and after the message arrives at its destination.
 Security can be selectively applied to different portions of a message (and to attachments if using XWSS).
 Message security can be used with intermediaries over multiple hops.

Question8: Explain various circumstances under which a message driven bean should be used.
Ans.

A message-driven bean is an enterprise bean that allows Java EE 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 Java EE component (an application client, another enterprise bean, or a web component) or by a JMS application or system that does not use Java EE technology. Message-driven beans

can process JMS messages or other kinds of messages. ***What Makes Message-Driven Beans Different from Session Beans?***

The most visible difference between message-driven beans and session 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 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, for example, 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.

Question9: Write a code in JSP to insert records in a student table with fields: student- ID, student-name, program, semester, student address using JDBC. Assume that the student table is created in database. Create records with the above fields in thee database.

Ans.

```
<%@page import="java.sql.*"%>
<%!
int stu_id=1;
String stu_name="Harshita ";
String program=”BBA”
String sem=”1”;
String address =”Bhopal”;
%>
<%

try{ Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

```
Connection con=DriverManager.getConnection("jdbc:odbc:ignousolver ");
Statement st=con.createStatement();
String query="insert into student
values("+stu_id+",'"+stu_name+"','"+program+","+sem+" , “ ,”+address+”)";

if(st.executeUpdate(query)>0){

out.write("Records inserted");

}else{

out.write("insertion faild");

} }catch(Exception e){

out.write("Exception : "+e); }

 %>
```

Second Record

```
%@page import="java.sql.*"%
 <%!
 int stu_id=2; String stu_name="Ishita "; String program=”BBA”
String sem=”1”;
String address =”Bhopal”;
%>
<%
try{
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
Connection con=DriverManager.getConnection("jdbc:odbc:ignousolver ");
Statement st=con.createStatement();
String query="insert into student
values("+stu_id+",'"+stu_name+"','"+program+","+sem+" , “ ,”+address+”)";
if(st.executeUpdate(query)>0){

out.write("Records inserted");

 }else{

out.write("insertion faild");
```

```
 }

}catch(Exception e){

out.write("Exception : "+e);

}

%>
```

Question10: Design a login page and write code for login button using JSP.

Ans.

LOGIN PAGE.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<link href="mainpage.css" rel="stylesheet"/>
<title>LIVE CHAT, LIFE CHAT</title>
<style>
.logindiv
{
top: 0px;
position: relative;
width:100%;
height: 300px;
background-color: white;
float: left;

}
.bodydiv1
{
position: relative;
top:150px;
}
.textclass
{
font-family: sans-serif;
font-size: 20;
```

```
}
.textclass:hover
{
border: solid;
border-top-color: blueviolet;
border-bottom-color: blueviolet;
}
.buttonclass {
padding: 15px 25px;
font-size: 20px;
text-align: center;
cursor: pointer;

outline: none;
color: #fff;
background-color: #4CAF50;
border: #336699;
border-radius: 15px;
box-shadow: 0 9px #999;
height: 40px;
}
.buttonclass:hover {background-color: #3e8e41}
.buttonclass:active {
background-color: #3e8e41;
box-shadow: 0 5px #666;
transform: translateY(4px);
}
</style>
</head>
<body>
<div class="bodydiv1">
<jsp:include page="WEB-INF/HEADERPAGE.jsp"></jsp:include>
<div id="logindiv" class="logindiv">
<form action ="TestSaveLogin.jsp" method ="post">

<center>
<table width =" 40%" border="0px">
<tr><td height="65px"></td></tr>
<tr><td width="150px" height="65px"><b>USER NAME</b></td>
```

```
<td colspan ="2" align="center"height="65px"><input type ="text" name
="username"class="textclass" height="20%"></td></tr>
<tr><td height="65px"><b>Password</b></td>
<td colspan="2" align ="center" height="65px"><input type="password" name
="password" class="textclass">
</td>
</tr>
<tr><td height="65px" colspan ="2" align ="center"><input type="Submit" value
="LOGIN" class="buttonclass"> </td>
</tr>
</table>
</center>
</form>

</div>
<div style="position:relative; float: right; "><jsp:include page="WEB-
INF/FOOTERPAGE.jsp"></jsp:include> </div>
</div>
</body>
</html>
```

LOGIN.java

```
public class LoginManager {
private static String
loginpage="LoginForm.jsp",homepage="home.jsp",logoutpage="logout.jsp",
activUserpage = "GetLoggedInUserList.jsp", Activitypage="ChatPage.jsp" ,
dashboard="userdashboard.jsp" , popupmenu="pop2.jsp";
public static String GetImage(HttpSession session)
{
try
{
String src=LoginManager.currentUser(session);
String rnd=getRandom();
String img= "<img style=' border-radius:50px; height:150px; width:200px;'
src='getpicexample.jsp?src="+src+".jpg&rnd=" + rnd + "' alt="/>" ;
return img;
}
catch(Exception ex)

{
System.out.println(ex);
```

```
return "";
}
public static boolean doLogin(HttpSession session,Object username,Object
password, HttpServletResponse response)
{
try
{
boolean result=LoginManager.isUsernameAndPasswordCorrect(session,
username, password);
if(!result)
{
response.sendRedirect(popupmenu);
return false;
}
session.setAttribute("user", username);
int no = LoginManager.saveLogin(username, session);
session.setAttribute("login_no", no);

response.sendRedirect(dashboard);
return true;
}
catch(Exception ex)
{
System.out.println(ex);
return false;
}
}
public static boolean isUsernameAndPasswordCorrect(HttpSession session,Object
username,Object password)
{
try
{
Connection con= ConnectionClass.connect(session);
PreparedStatement ps=con.prepareStatement("select * from chat_users where
user_name=? and password=?");
ps.setString(1, "" + username);
ps.setString(2,"" + password);

ResultSet rs=ps.executeQuery();
if(rs.next())
```

```
return true;
return false;
}
catch(Exception ex)
{
System.out.println(ex);
return false;
}
}
}
```

Question11: Crate a database of 10 records in customer tables with field (customer-ID, customer-name, customer- phone, customer- address) in Oracle Write a program using Sevelet and JDBC that will display all the records of the customer in ascending order of customer-ID.

Ans.

STEP 1. Import required packages

```
import java.sql.*;

public class JDBCExample {
// JDBC driver name and database URL
static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
static final String DB_URL = "jdbc:mysql://localhost/Customers";
// Database credentials
static final String USER = "customarname";
static final String PASS = "password";
public static void main(String[] args) {
Connection conn = null;
Statement stmt = null;
try{
//STEP 2: Register JDBC driver
Class.forName("com.mysql.jdbc.Driver");
//STEP 3: Open a connection
System.out.println("Connecting to a selected database...");
conn = DriverManager.getConnection(DB_URL, USER, PASS);
System.out.println("Connected database successfully...");
//STEP 4: Execute a query
System.out.println("Creating statement...");
```

```
stmt = conn.createStatement();
// Extract records in ascending order by first name.
System.out.println("Fetching records in ascending order...");
String sql = "SELECT Cust_ id, Cust_name, Cust_phone, Cust_addr FROM
Registration" +
"ORDER BY first ASC";
ResultSet rs = stmt.executeQuery(sql);
while(rs.next()){
//Retrieve by column name
int Cust_id = rs.getInt("id");
String Cust_name = rs.getString("name");
int Cust_phone = rs.getInt("mob");
String Cust_aadr = rs.getString("addr");
//Display values
System.out.print("ID: " + id);
System.out.print(", Name: " + name);
System.out.print(", Phone no: " + mob);
System.out.println(", Address: " + addr);
}
rs.close();

}
catch(SQLException se){
//Handle errors for JDBC
se.printStackTrace();
}catch(Exception e){
//Handle errors for Class.forName
e.printStackTrace();
}finally{
//finally block used to close resources
try{
if(stmt!=null)
conn.close();
}catch(SQLException se){
}// do nothing
try{
if(conn!=null)
conn.close();
}catch(SQLException se){
se.printStackTrace();
```

```
}//end finally try
}//end try
System.out.println("Goodbye!");
}//end main

}//end JDBCExample
```