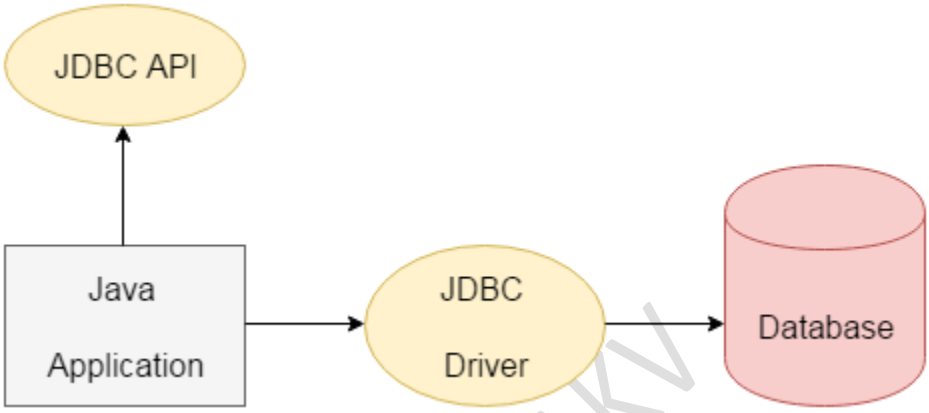
Unit V Part B JDBC Tutorial

JDBC stands for Java Database Connectivity. JDBC is a Java API to connect and execute the query with the database. It is a part of JavaSE (Java Standard Edition). JDBC API uses JDBC drivers to connect with the database. There are four types of JDBC drivers:

- JDBC-ODBC Bridge Driver,
- Native Driver,
- Network Protocol Driver, and
- Thin Driver

We can use JDBC API to access tabular data stored in any relational database. By the help of JDBC API, we can save, update, delete and fetch data from the database. It is like Open Database Connectivity (ODBC) provided by Microsoft.



The current version of JDBC is 4.3. It is the stable release since 21st September, 2017. The **java.sql** package contains classes and interfaces for JDBC API. A list of popular *interfaces* of JDBC API are given below:

- Driver interface
- Connection interface
- Statement interface
- PreparedStatement interface
- CallableStatement interface
- ResultSet interface
- ResultSetMetaData interface
- DatabaseMetaData interface
- RowSet interface

A list of popular *classes* of JDBC API are given below:

- DriverManager class
- Blob class
- Clob class
- Types class

Why Should We Use JDBC

Before JDBC, ODBC API was the database API to connect and execute the query with the database. But, ODBC API uses ODBC driver which is written in C language (i.e. platform

dependent and unsecured). That is why Java has defined its own API (JDBC API) that uses JDBC drivers (written in Java language).

We can use JDBC API to handle database using Java program and can perform the following activities:

1. Connect to the database
2. Execute queries and update statements to the database
3. Retrieve the result received from the database.

What is API

API (Application programming interface) is a document that contains a description of all the features of a product or software. It represents classes and interfaces that software programs can follow to communicate with each other. An API can be created for applications, libraries, operating systems, etc.

JDBC Driver

JDBC Driver is a software component that enables java application to interact with the database.

There are 4 types of JDBC drivers:

1. JDBC-ODBC bridge driver
2. Native-API driver (partially java driver)
3. Network Protocol driver (fully java driver)
4. Thin driver (fully java driver)

1) JDBC-ODBC bridge driver

The JDBC-ODBC bridge driver uses ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls. This is now discouraged because of thin driver.

Oracle does not support the JDBC-ODBC Bridge from Java 8. Oracle recommends that you use JDBC drivers provided by the vendor of your database instead of the JDBC-ODBC Bridge.

Advantages:

- easy to use.
- can be easily connected to any database.

Disadvantages:

- Performance degraded because JDBC method call is converted into the ODBC function calls.
- The ODBC driver needs to be installed on the client machine.

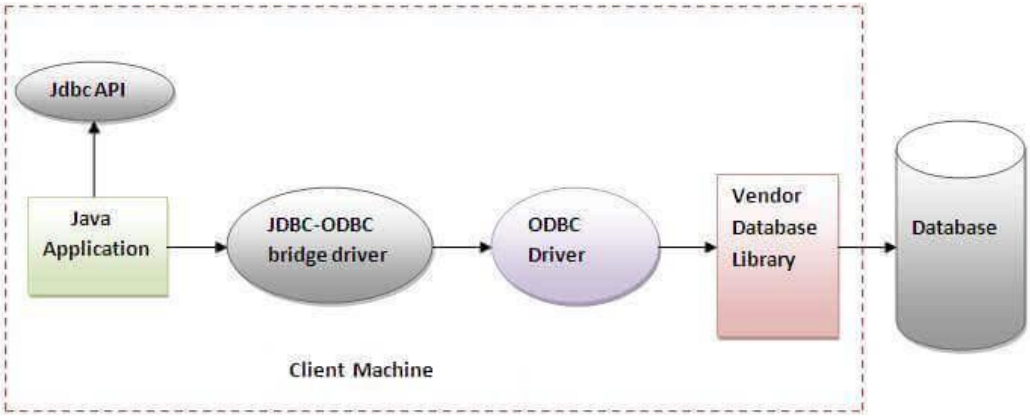


Figure- JDBC-ODBC Bridge Driver

2) Native-API driver

The Native API driver uses the client-side libraries of the database. The driver converts JDBC method calls into native calls of the database API. It is not written entirely in java.

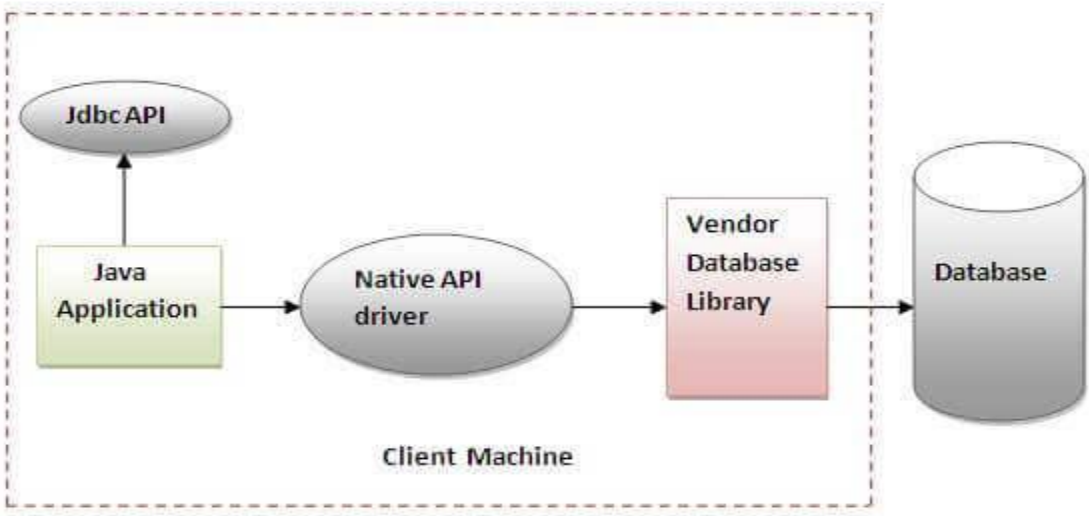


Figure- Native API Driver

Advantage:

- performance upgraded than JDBC-ODBC bridge driver.

Disadvantage:

- The Native driver needs to be installed on the each client machine.
- The Vendor client library needs to be installed on client machine.

3) Network Protocol driver

The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. It is fully written in java.

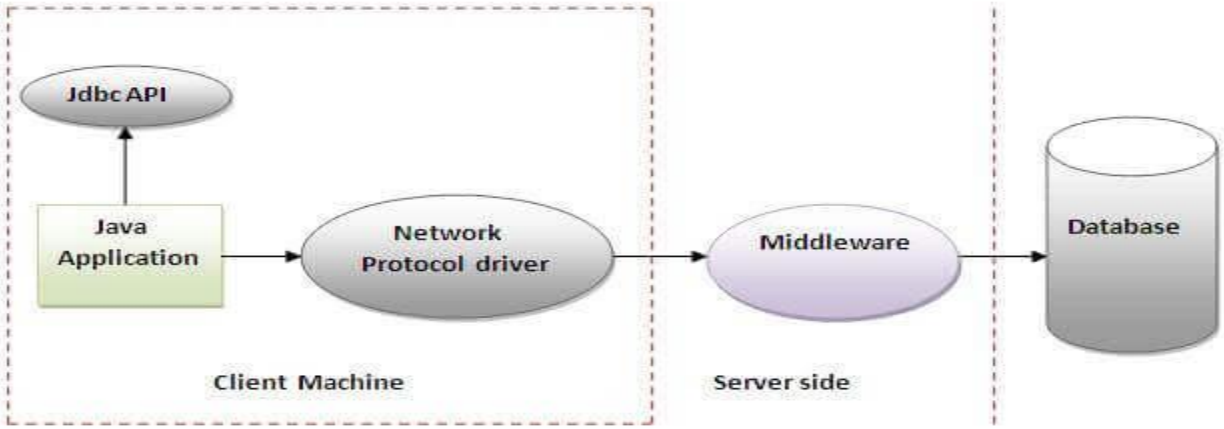


Figure- Network Protocol Driver

Advantage:

- No client side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.

Disadvantages:

- Network support is required on client machine.
- Requires database-specific coding to be done in the middle tier.
- Maintenance of Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier.

4) Thin driver

The thin driver converts JDBC calls directly into the vendor-specific database protocol. That is why it is known as thin driver. It is fully written in Java language.

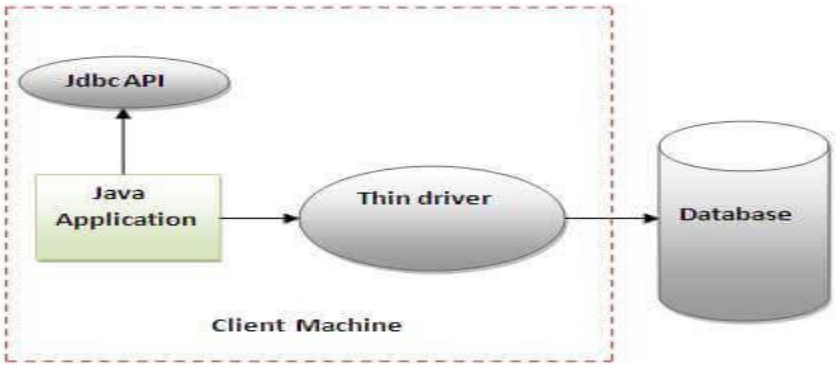


Figure- Thin Driver

Advantage:

- Better performance than all other drivers.
- No software is required at client side or server side.

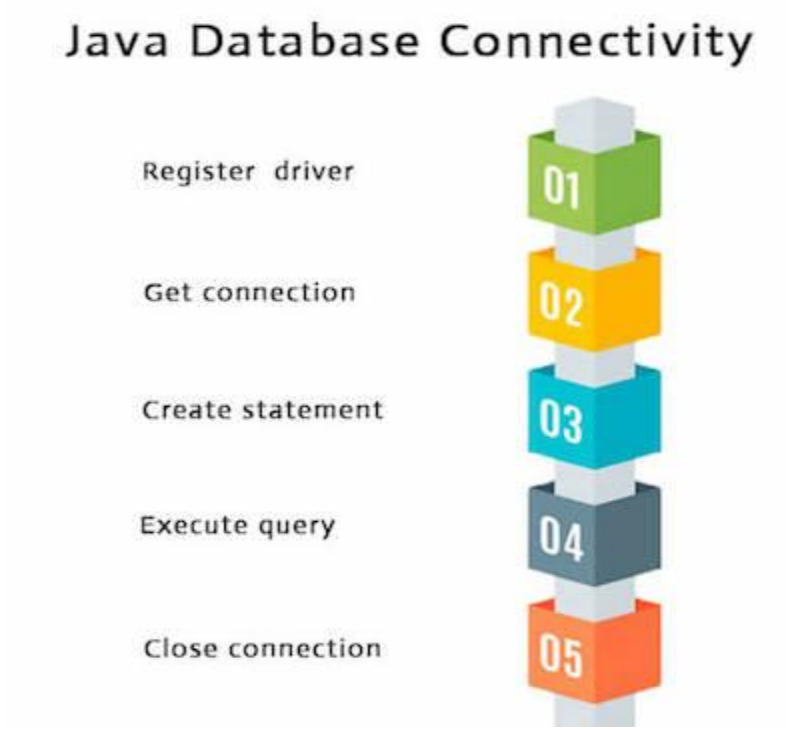
Disadvantage:

- Drivers depend on the Database.

Java Database Connectivity with 5 Steps

There are 5 steps to connect any java application with the database using JDBC. These steps are as follows:

- Register the Driver class
- Create connection
- Create statement
- Execute queries
- Close connection



1) Register the driver class

The **forName()** method of Class class is used to register the driver class. This method is used to dynamically load the driver class.

Syntax of forName() method

public static void forName(String className)throws ClassNotFoundException

Example to register the OracleDriver class

Here, Java program is loading oracle driver to esteblish database connection.

```
Class.forName("oracle.jdbc.driver.OracleDriver");
```

2) Create the connection object

The **getConnection()** method of DriverManager class is used to establish connection with the database.

Syntax of getConnection() method

1. 1) public static Connection getConnection(String url)throws SQLException
2. 2) public static Connection getConnection(String url,String name,String password)

3. throws SQLException

Example to establish connection with the Oracle database

```
Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","password");
```

3) Create the Statement object

The createStatement() method of Connection interface is used to create statement. The object of statement is responsible to execute queries with the database.

Syntax of createStatement() method

public Statement createStatement()throws SQLException

Example to create the statement object

```
Statement stmt=con.createStatement();
```

4) Execute the query

The executeQuery() method of Statement interface is used to execute queries to the database. This method returns the object of ResultSet that can be used to get all the records of a table.

Syntax of executeQuery() method

1. public ResultSet executeQuery(String sql)throws SQLException

Example to execute query

```
ResultSet rs=stmt.executeQuery("select * from emp");
while(rs.next())
{
    System.out.println(rs.getInt(1)+" "+rs.getString(2));
}
```


5) Close the connection object

By closing connection object statement and ResultSet will be closed automatically. The close() method of Connection interface is used to close the connection.

Syntax of close() method

1. public void close()throws SQLException

Example to close connection

```
con.close();
```

Java Database Connectivity with MySQL

To connect Java application with the MySQL database, we need to follow 5 following steps.
In this example we are using MySql as the database. So we need to know following informations for the mysql database:

1. **Driver class:** The driver class for the mysql database is **com.mysql.jdbc.Driver**.
2. **Connection URL:** The connection URL for the mysql database is **jdbc:mysql://localhost:3306/org** where jdbc is the API, mysql is the database, localhost is the server name on which mysql is running, we may also use IP address, 3306 is the port number and org is the database name. We may use any database, in such case, we need to replace the org with our database name.
3. **Username:** The default username for the mysql database is **root**.

4. **Password:** It is the password given by the user at the time of installing the mysql database. In this example, we are going to use 123 as the password.

Let's first create a table in the mysql database, but before creating table, we need to create database first.

1. **create database org;**
2. **use org;**
3. **create table emp(id int(10),name varchar(40),age int(3));**

Example to Connect Java Application with mysql database

In this example, org is the database name, root is the username and password is 123.

```
import java.sql.*;
class MysqlCon
{
   public static void main(String args[])
        {
                try
                {
                        Class.forName("com.mysql.jdbc.Driver");
                        Connection
con=DriverManager.getConnection("jdbc:mysql://localhost:3306/org","root","123");
                        //here org is database name, root is username and 123 is password
                        Statement stmt=con.createStatement();
                        ResultSet rs=stmt.executeQuery("select * from emp");
                        while(rs.next())
                        System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"
"+rs.getInt(3));
                        con.close();
                }catch(Exception e)
                {
                        System.out.println(e);
                }
   }
}
```

To connect java application with the mysql database, **mysqlconnector.jar** file is required to be loaded.

download the jar file mysql-connector.jar

Two ways to load the jar file:

1. Paste the mysqlconnector.jar file in jre/lib/ext folder
2. Set classpath

1) Paste the mysqlconnector.jar file in JRE/lib/ext folder:

Download the mysqlconnector.jar file. Go to jre/lib/ext folder and paste the jar file here.

2) Set classpath:

There are two ways to set the classpath:

- temporary
- permanent

How to set the temporary classpath

open command prompt and write:

1. C:>set classpath=c:\folder\mysql-connector-java-5.0.8-bin.jar;

How to set the permanent classpath

Go to environment variable then click on new tab. In variable name write **classpath** and in variable value paste the path to the mysqlconnector.jar file by appending mysqlconnector.jar; as C:\folder\mysql-connector-java-5.0.8-bin.jar;.;

DriverManager class

The DriverManager class acts as an interface between user and drivers. It keeps track of the drivers that are available and handles establishing a connection between a database and the appropriate driver. The DriverManager class maintains a list of Driver classes that have registered themselves by calling the method DriverManager.registerDriver().

Useful methods of DriverManager class

Method	Description
1) public static void registerDriver(Driver driver):	is used to register the given driver with DriverManager.
2) public static void deregisterDriver(Driver driver):	is used to deregister the given driver (drop the driver from the list) with DriverManager.
3) public static Connection getConnection(String url):	is used to establish the connection with the specified url.
4) public static Connection getConnection(String url,String userName,String password):	is used to establish the connection with the specified url, username and password.

Connection interface

A Connection is the session between java application and database. The Connection interface is a factory of Statement, PreparedStatement, and DatabaseMetaData i.e. object of Connection can be used to get the object of Statement and DatabaseMetaData. The Connection interface provide many methods for transaction management like commit(), rollback() etc.

Commonly used methods of Connection interface:

1) public Statement createStatement(): creates a statement object that can be used to execute SQL queries.

2) public Statement createStatement(int resultSetType,int resultSetConcurrency): Creates a Statement object that will generate ResultSet objects with the given type and concurrency.

3) public void setAutoCommit(boolean status): is used to set the commit status.By default it is true.

4) public void commit(): saves the changes made since the previous commit/rollback permanent.

5) public void rollback(): Drops all changes made since the previous commit/rollback.

6) public void close(): closes the connection and Releases a JDBC resources immediately.

Statement interface

The **Statement interface** provides methods to execute queries with the database. The statement interface is a factory of ResultSet i.e. it provides factory method to get the object of ResultSet.

Commonly used methods of Statement interface:

The important methods of Statement interface are as follows:

1) public ResultSet executeQuery(String sql): is used to execute SELECT query. It returns the object of ResultSet.

2) public int executeUpdate(String sql): is used to execute specified query, it may be create, drop, insert, update, delete etc.

3) public boolean execute(String sql): is used to execute queries that may return multiple results.

4) public int[] executeBatch(): is used to execute batch of commands.

Example of Statement interface

Let's see the simple example of Statement interface to insert, update and delete the record.

```
import java.sql.*;
class FetchRecord
{
        public static void main(String args[])throws Exception
        {
                Class.forName("com.mysql.jdbc.Driver");
                Connection con=
DriverManager.getConnection("jdbc:mysql://localhost:3306/org","root","123");
                Statement stmt=con.createStatement();
                //stmt.executeUpdate("insert into emp values(103,'Irfan',32)");
                //int result=stmt.executeUpdate("update emp set name='rajesh',age=40
where id=103");
                int result=stmt.executeUpdate("delete from emp where id=103");
                System.out.println(result+" records affected");
                con.close();
        }
}
```

ResultSet interface

The object of ResultSet maintains a cursor pointing to a row of a table. Initially, cursor points to before the first row. But we can make this object to move forward and backward direction by passing either TYPE_SCROLL_INSENSITIVE or TYPE_SCROLL_SENSITIVE in createStatement(int,int) method as well as we can make this object as updatable by:

1. Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_UPDATABLE);

Commonly used methods of ResultSet interface

1) public boolean next():	is used to move the cursor to the one row next from the current position.
2) public boolean previous():	is used to move the cursor to the one row previous from the current position.
3) public boolean first():	is used to move the cursor to the first row in result set object.
4) public boolean last():	is used to move the cursor to the last row in result set object.
5) public boolean absolute(int row):	is used to move the cursor to the specified row number in the ResultSet object.
6) public boolean relative(int row):	is used to move the cursor to the relative row number in the ResultSet object, it may be positive or negative.
7) public int getInt(int columnIndex):	is used to return the data of specified column index of the current row as int.
8) public int getInt(String columnName):	is used to return the data of specified column name of the current row as int.
9) public String getString(int columnIndex):	is used to return the data of specified column index of the current row as String.
10) public String getString(String columnName):	is used to return the data of specified column name of the current row as String.

```
import java.sql.*;
class FetchRecord1
{
	public static void main(String args[])throws Exception
	{
		Class.forName("com.mysql.jdbc.Driver");
		Connection con=
DriverManager.getConnection("jdbc:mysql://localhost:3306/org","root","123");
		Statement stmt=
con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPDA
TABLE);
		ResultSet rs=stmt.executeQuery("select * from emp");
		//getting the record of 2rd row
		rs.absolute(2);
		System.out.println(rs.getString(1)+" "+rs.getString(2)+" "+rs.getString(3));
		con.close();
	}
}
```

PreparedStatement interface

The PreparedStatement interface is a subinterface of Statement. It is used to execute parameterized query.

Let's see the example of parameterized query:

1. String sql="insert into emp values(?,?,?)";

As you can see, we are passing parameter (?) for the values. Its value will be set by calling the setter methods of PreparedStatement.

Why use PreparedStatement?

Improves performance: The performance of the application will be faster if you use PreparedStatement interface because query is compiled only once.

Methods of PreparedStatement interface

The important methods of PreparedStatement interface are given below:

Method	Description
public void setInt(int paramIndex, int value)	sets the integer value to the given parameter index.
public void setString(int paramIndex, String value)	sets the String value to the given parameter index.
public void setFloat(int paramIndex, float value)	sets the float value to the given parameter index.
public void setDouble(int paramIndex, double value)	sets the double value to the given parameter index.
public int executeUpdate()	executes the query. It is used for create, drop, insert, update, delete etc.
public ResultSet executeQuery()	executes the select query. It returns an instance of ResultSet.

Example of PreparedStatement interface that inserts the record

```
import java.sql.*;
import java.io.*;
class RS
{
   public static void main(String args[])throws Exception
        {
                Class.forName("com.mysql.jdbc.Driver");
                Connection
con=DriverManager.getConnection("jdbc:mysql://localhost:3306/org","root","123");
      PreparedStatement ps=con.prepareStatement("insert into emp values(?,?,?)");
      BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
      do
                {
                        System.out.println("enter id:");
                        int id=Integer.parseInt(br.readLine());
```

```
                    System.out.println("enter name:");
                    String name=br.readLine();
                    System.out.println("enter age:");
                    int age=Integer.parseInt(br.readLine());
                    ps.setInt(1,id);
                    ps.setString(2,name);
                    ps.setInt(3,age);
                    int i=ps.executeUpdate();
                    System.out.println(i+" records affected");
                System.out.println("Do you want to continue: y/n");
                    String s=br.readLine();
                    if(s.startsWith("n"))
                    {
                        break;
                    }
            }while(true);
            con.close();
    }
}
```

Example to connect JFrame to My Sql


```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.sql.*;
import javax.swing.table.*;
class EmpLogin extends JFrame implements ActionListener
{
    JLabel l1,l2;
    JTextField t1;
    JPasswordField p;
    JButton b1,b2;
    static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
    static final String connectionUrl = "jdbc:mysql://localhost/emp1";
    static final String dbUser = "root";
    static final String dbPwd = "123";
    static Connection conn;
    EmpLogin()
    {
        l1=new JLabel("User Id");
        l1.setBounds(50,50,100,30);
        l2=new JLabel("Password");
        l2.setBounds(50,100,100,30);
```

```
            t1=new JTextField("amit");
            t1.setBounds(150,50,200,30);
            p=new JPasswordField("123");
            p.setBounds(150,100,200,30);
            b1=new JButton("Login");
            b1.setBounds(75,175,95,30);
            b2=new JButton("Cancel");
            b2.setBounds(220,175,95,30);
            b1.addActionListener(this);
            b2.addActionListener(new ActionListener()
            {
                  public void actionPerformed(ActionEvent e)
                  {
                        dispose();
                  }
            });
            add(l1);
            add(l2);
            add(t1);
            add(p);
            add(b1);
            add(b2);
            setSize(400,300);
            setLayout(null);
            setVisible(true);
      }
      public void actionPerformed(ActionEvent e)
      {
            String user,pass;
            user=t1.getText();
            pass=p.getText();
            String queryString = "SELECT user1,pass1 FROM emp";
            try
            {
                  //Class.forName(JDBC_DRIVER);
                  conn = DriverManager.getConnection(connectionUrl, dbUser,
dbPwd);
                  Statement stmt = conn.createStatement();
                  ResultSet rs;
                  rs = stmt.executeQuery(queryString);
                  rs.next();
                  if((user.equals(rs.getString(1))) && (pass.equals(rs.getString(2))))
```

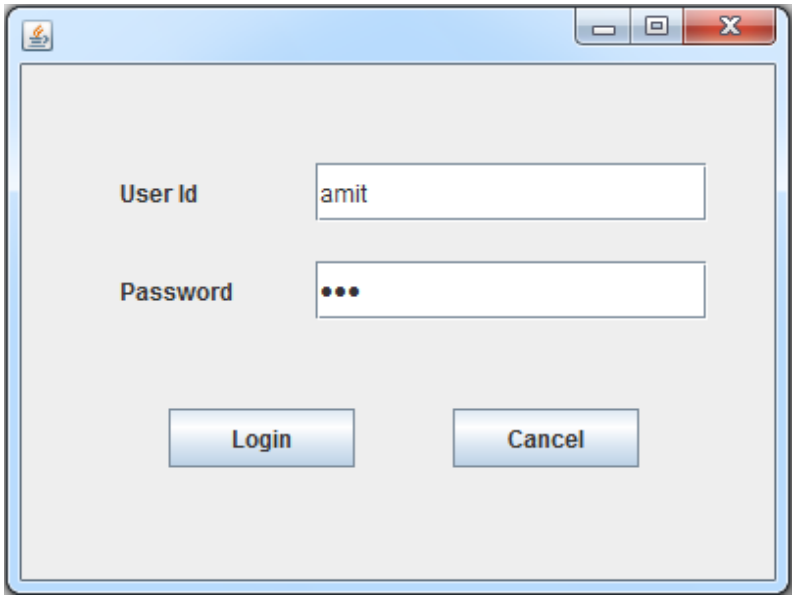
```
                {
                        EmpRegister er=new EmpRegister();
                }
                else
                {
                        JOptionPane.showMessageDialog(null,"Incorrect Login");
                }
            }
            catch (SQLException sqle)
            {
                System.out.println("SQL Exception thrown: " + sqle);
            }
        }
}
class EP1
{
        public static void main(String args[])
        {
            EmpLogin ep=new EmpLogin();
            ep.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        }
}
class EmpRegister extends JFrame implements ActionListener
{
        JLabel l3,l4,l5,l6,l7,l8,l9,l10;
        JTextField t2,t3,t4,t5,t6,t7;
        JButton b3;
        JRadioButton r1,r2;
        static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
        static final String connectionUrl = "jdbc:mysql://localhost/emp1";
        static final String dbUser = "root";
        static final String dbPwd = "123";
        static Connection conn;
        String columnNames[]=
{"eid","ename","edes","egender","eaddress","email","ephone"};
        EmpRegister()
        {
            l3=new JLabel("REGISTRATION FORM");
            l3.setBounds(150,10,150,30);
            l4=new JLabel("Emp Id");
            l4.setBounds(50,50,100,30);
            l5=new JLabel("Emp Name");
```

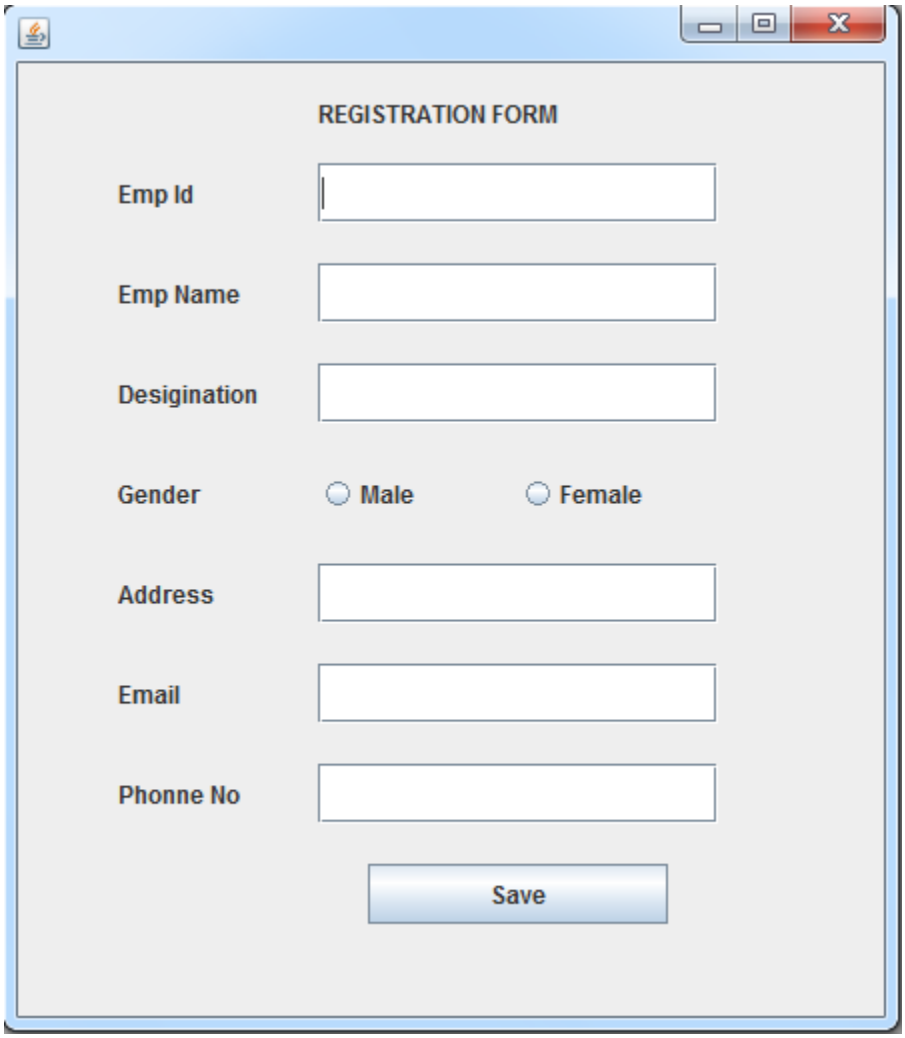
```
l5.setBounds(50,100,100,30);
l6=new JLabel("Desigination");
l6.setBounds(50,150,100,30);
l10=new JLabel("Gender");
l10.setBounds(50,200,100,30);
l7=new JLabel("Address");
l7.setBounds(50,250,100,30);
l8=new JLabel("Email");
l8.setBounds(50,300,100,30);
l9=new JLabel("Phonne No");
l9.setBounds(50,350,100,30);
t2=new JTextField();
t2.setBounds(150,50,200,30);
t3=new JTextField();
t3.setBounds(150,100,200,30);
t4=new JTextField();
t4.setBounds(150,150,200,30);
r1=new JRadioButton("Male");
r1.setBounds(150,200,90,30);
r2=new JRadioButton("Female");
r2.setBounds(250,200,90,30);
ButtonGroup bg=new ButtonGroup();
bg.add(r1);
bg.add(r2);
t5=new JTextField();
t5.setBounds(150,250,200,30);
t6=new JTextField();
t6.setBounds(150,300,200,30);
t7=new JTextField();
t7.setBounds(150,350,200,30);
b3=new JButton("Save");
b3.setBounds(175,400,150,30);
add(l3);
add(l4);
add(l5);
add(l6);
add(l10);
add(l7);
add(l8);
add(l9);
add(t2);
add(t3);
```

```
            add(t4);
            add(r1);
            add(r2);
            add(t5);
            add(t6);
            add(t7);
            add(b3);
            setSize(600,600);
            setLayout(null);
            setVisible(true);
            b3.addActionListener(this);
        }
        public void actionPerformed(ActionEvent e)
        {
            String eid,ename,edes,egender,eaddress,email,ephone;
            eid=t2.getText();
            ename=t3.getText();
            edes=t4.getText();
            egender="Select";
            if(r1.isSelected())
            {
                egender="Male";
            }
            if(r2.isSelected())
            {
                egender="Female";
            }
            eaddress=t5.getText();
            email=t6.getText();
            ephone=t7.getText();
            String queryString = "Insert into
empreg(eid1,ename1,edes1,egender1,eadress1,email1,ephone)values(?,?,?,?,?,?,?)";
            try
            {
                conn = DriverManager.getConnection(connectionUrl, dbUser,
dbPwd);
                PreparedStatement ps=conn.prepareStatement(queryString);
                ps.setString(1, eid);
                ps.setString(2, ename);
                ps.setString(3, edes);
                ps.setString(4, egender);
                ps.setString(5, eaddress);
```

```
                    ps.setString(6, email);
                    ps.setString(7, ephone);
                    ps.executeUpdate();
                }
                catch (SQLException sqle)
                {
                    System.out.println("SQL Exception thrown: " + sqle);
                }
                JOptionPane.showMessageDialog(null, " Record Updated!");
        }
}
```

Output:





JAR Files

- Jar files, or Java Archive Files, are a platform independent form of zipped/compressed files. They come in two varieties,
    - **Standard Jar Files** - hold any files you want to compress together.
    - **Executable Jar Files** - hold .class, .gif, .jpg, files needed for a particular application. These files can be placed on your desktop and executed like .EXE files.
- **Steps to create an Executable JAR file in NETBEANS:**
        - Run your program at least once to be sure the main class is set
            - in the projects window, right click on the project, select "Properties" menu
            - under categories, select "run"
            - verify the main class is set correctly.
        - Under the **Run** menu item, select **Clean and Build Main Project**
        - Your Jar file will be created in your project folder under the **dist** sub folder.
    - Other notes:

- You can open a Jar file using Winzip, and execute a Jar file from your desktop by doubling clicking on it (You need to have the Java runtime environment installed).
- Many students have downloaded and installed unzip utilities that take control of the JAR extension (e.g. WinRAR), disabling the ability to execute the JAR by doubling clicking on it. You'll need to change the JAR file type (under Folder Options in Windows My-Computer Tools) to use JavaW to open the file.

Jar files in Java

A JAR (Java Archive) is a package file format typically used to aggregate many Java class files and associated metadata and resources (text, images, etc.) into one file to distribute application software or libraries on the Java platform.

In simple words, a JAR file is a file that contains compressed version of .class files, audio files, image files or directories. We can imagine a .jar files as a zipped file(.zip) that is created by using WinZip software. Even , WinZip software can be used to used to extract the contents of a .jar . So you can use them for tasks such as lossless data compression, archiving, decompression, and archive unpacking.

Let us see how to create a .jar file and related commands which help us to work with .jar files

1. **Create a JAR file:** To create a .jar file , we can use jar cf command in the following way:
   jar cf jarfilename inputfiles
   Here, cf represents create the file. For example , assuming our package pack is available in C:\directory , to convert it into a jar file into the pack.jar , we can give the command as:

 C:\> jar cf pack.jar pack

Now , pack.jar file is created

 Viewing a JAR file: To view the contents of .jar files, we can use the command as:

jar tf jarfilename

Here , tf represents table view of file contents. For example, to view the contents of our pack.jar file , we can give the command:

C:/> jar tf pack.jar

Now , the contents of pack.jar are displayed as:

META-INF/
META-INF/MANIFEST.MF
pack/
pack/class1.class
pack/class2.class
..
..

where class1 , class2 etc are the classes in the package pack. The first two entries represent that there is a manifest file created and added to pack.jar. The third entry represents the sub-directory with the name pack and the last two represent the files name in the directory pack.

When we create .jar files , it automatically receives the default manifest file. There can be only one manifest file in an archive , and it always has the pathname.
META-INF/MANIFEST.MF
This manifest file is useful to specify the information about other files which are packaged.

 Extracting a JAR file: To extract the files from a .jar file , we can use:
jar xf jarfilename
Here, xf represents extract files from the jar files. For example , to extract the contents of our pack.jar file, we can write:
C:\> jar xf pack.jar
This will create the following directories in C:\
META-INF
pack // in this directory , we can see class1.class and class2.class.

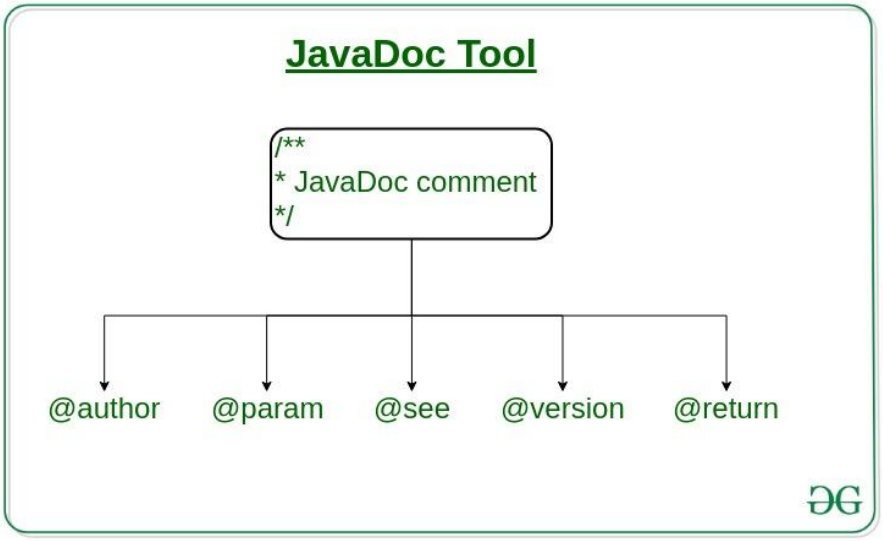
 Updating a JAR File The Jar tool provides a 'u' option which you can use to update the contents of an existing JAR file by modifying its manifest or by adding files. The basic command for adding files has this format:
jar uf jar-file input-file(s)
here uf represent update jar file. For example , to update the contents of our pack.jar file, we can write:
C:\>jar uf pack.jar

 Running a JAR file: In order to run an application packaged as a JAR file (requires the Main-class manifest header) , following command can be used:
C:\>java -jar pack.jar

What is JavaDoc tool and how to use it?

JavaDoc tool is a **document generator tool** in Java programming language for generating standard documentation in HTML format. It generates API documentation. It parses the declarations ad documentation in a set of source file describing classes, methods, constructors and fields.

Before using JavaDoc tool, you must include JavaDoc comments /**……..*/ providing information about classes, methods and constructors etc. For creating a good and understandable document API for any java file you must write better comments for every class, method, constructor.



The JavaDoc comments is different from the normal comments because of the extra asterisk at the beginning of the comment. It may contain the **HTML tags** as well.

// Single-Line Comment

/*
* Multiple-Line comment
*/

/
*** JavaDoc comment**
***/**

By writing a number of comments, it **does not affect the performance** of the Java program as all the comments are removed at compile time.

JavaDoc Format: –
It has two parts: – a description which is followed by block tags.

Some Integrated Development Environments (IDE) automatically generate the JavaDoc file like NetBeans, IntelliJ IDEA, Eclipse etc.

Generation of JavaDoc: –
To create a JavaDoc you do not need to compile the java file. To create the Java documentation API, you need to write Javadoc followed by file name.

javadoc file_name or javadoc package_name

After successful execution of the above command, a number of HTML files will be created, open the file named index to see all the information about classes.

JavaDoc Tags

Tag	Parameter	Description
@author	author_name	Describes an author
@param	desciption	provide information about method parameter or the input it takes
@see	reference	generate a link to other element of the document
@version	version-name	provide version of the class, interface or enum.
@return	desciption	provide the return value

Example 1: –

```
package exa;
import java.util.Scanner;

/**
*
* @author Yash
*/
public class Example {
   /**
* This is a program for adding two numbers in java.
   * @param args
*/
   public static void main(String[] args)
   {
      /**
      * This is the main method
      * which is very important for
      * execution for a java program.
      */

      int x, y;
      Scanner sc = new Scanner(System.in);
      /**
      * Declared two variables x and y.
      * And taking input from the user
      * by using Scanner class.
      *
      */

      x = sc.nextInt();
      y = sc.nextInt();
      /**
      * Storing the result in variable sum
      * which is of the integer type.
```

```
        */
        int sum = x + y;

        /**
        * Using standard output stream
        * for giving the output.
        * @return null
        */
        System.out.println("Sum is: " + sum);
    }
}
```

Generating document for the above class

javadoc exa