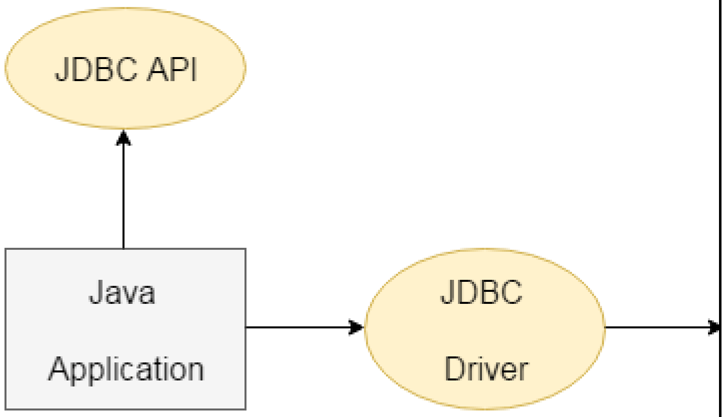
JDBC

- 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.



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

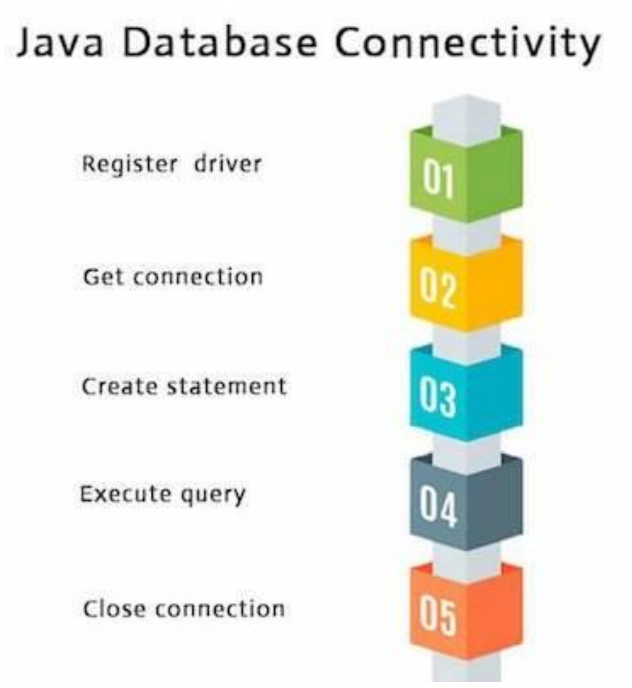
A list of popular classes of JDBC API are given below:

- DriverManager class
- Types class

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



Steps to Connect to Database

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.

Example to register the OracleDriver class

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

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

2. **Create the connection object**

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

Example

```
Connection con=DriverManager.getConnection(
    "jdbc:oracle:thin:@//localhost:1521/orcl",
    "username",
    "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.

Example

```
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.

Example

```
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

Example

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

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.

<u>Commonly used methods of Connection</u>

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

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

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

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

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

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.

<u>Commonly used methods of Statement interface:</u>

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

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

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

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

ResultSet Interface

- The object of ResultSet maintains a cursor pointing to a row of a table. Initially, cursor points to before the first row.
- By default, ResultSet object can be moved forward only and it is not updatable.

Commonly used methods of ResultSet interface

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.

public boolean last(): is used to move the cursor to the last row in result set object.

public int getInt(int columnIndex): is used to return the data of specified column index of the current row as int.

public int getInt(String columnName): is used to return the data of specified column name of the current row as int.

public String getString(int columnIndex): is used to return the data of specified column index of the current row as String.

public String getString(String columnName): is used to return the data of specified column name of the current row as String.

PreparedStatement interface

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

```
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.

Methods of PreparedStatement interface

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.

ResultSetMetaData Interface

- The metadata means data about data i.e. we can get further information from the data.
- If you have to get metadata of a table like total number of column, column name, column type etc. ,
- ResultSetMetaData interface is useful because it provides methods to get metadata from the ResultSet object.

Commonly used methods of ResultSetMetaData

public int getColumnCount() : it returns the total number of columns in the ResultSet object.

public String getColumnName(int index): it returns the column name of the specified column index.

public String getColumnTypeName(int index): it returns the column type name for the specified index.

public String getTableName(int index): it returns the table name for the specified column index.

Example to SELECT Data from Oracle

```
import java.sql.*;
class Demo3{
        public static void main(String[] args) throws
Exception {
   Class.forName("oracle.jdbc.driver.OracleDriver");
  System.out.println("Driver Loaded Successfully.");
  Connection con = DriverManager.getConnection(
       "jdbc:oracle:thin://@LOCALHOST:1521/orcl",
        "scott", "tiger");
        System.out.println("Connection Estblished");

        Statement s = con.createStatement();
        ResultSet rs = s.executeQuery("SELECT
DNAME, ROUND(AVG(SAL),2), MAX(SAL), MIN(SAL)
FROM EMP E JOIN DEPT D
ON(E.DEPTNO=D.DEPTNO) GROUP BY DNAME ");

    while(rs.next())
  System.out.printf("%-10s%10s%10s%10s\n",
              rs.getString(1),rs.getString(2),
              rs.getString(3),rs.getString(4));
        }
}
```

To Insert Record

```
  PreparedStatement ps = con.prepareStatement("
              INSERT INTO EMP(EMPNO,ENAME,SAL)
                        VALUES(?,?,?)");
   ps.setInt(1,9999);
   ps.setString(2,"Anand Tank");
   ps.setInt(3,52000);
   ps.executeUpdate();
```

To Update Record

```
PreparedStatement ps = con.prepareStatement(
UPDATE EMP SET ENAME=?,SAL=? WHERE EMPNO=?");
ps.setInt(3,9999);
ps.setString(1,"NAMAN MEHTA");
ps.setInt(2,25000);
ps.executeUpdate();
```

To Delete Record

```
PreparedStatement ps = con.prepareStatement(
"DELETE FROM EMP WHERE EMPNO = ?");
ps.setInt(1,9999);
ps.executeUpdate();
```

JDBC (Menu Driven – INSERT, UPDATE, DELETE, SEARCH)

```
import java.sql.*;
class DBConnection{
        Connection con;
        PreparedStatement ps;
        ResultSet rs;
        DBConnection(){
                try{

                        Class.forName("oracle.jdbc.driver.OracleDriver");
                        con = DriverManager.getConnection("jdbc:oracle:thin:@10.9.150.16:1521/atmiyadb",
                                                "c##22mca10","m10");
                        System.out.println("Connection Established");
                }catch(Exception e){}
        }

        public void insert(int ccode, String cname, int score) throws Exception{
          ps = con.prepareStatement("INSERT INTO CRICKETER(CCODE,CNAME,SCORE) VALUES(?,?,?)");
                ps.setInt(1,ccode);
                ps.setString(2,cname);
                ps.setInt(3,score);
                ps.executeUpdate();
        }

    public void update(int ccode, String cname, int score) throws Exception{
      ps = con.prepareStatement("UPDATE CRICKETER SET CNAME = ?, SCORE = ? WHERE CCODE = ?");
                ps.setInt(3,ccode);
                ps.setString(1,cname);
                ps.setInt(2,score);
                ps.executeUpdate();
        }

     public void delete(int ccode) throws Exception{
                ps = con.prepareStatement("SELECT * FROM CRICKETER WHERE CCODE = ?");
                ps.setInt(1,ccode);
                rs = ps.executeQuery();

                if(rs.next()){
                        ps = con.prepareStatement("DELETE FROM CRICKETER WHERE CCODE = ?");
                        ps.setInt(1,ccode);
                        ps.executeUpdate();
                }
                else
                        System.out.println("Record Not Found ....!");
        }
        public void display() throws Exception{
                ps = con.prepareStatement("SELECT * FROM CRICKETER");
                rs = ps.executeQuery();
```

```
            while(rs.next()){
              System.out.printf("%-15s%-20s%5s\n",rs.getString(1),rs.getString(2),rs.getString(3));
            }
    }

    public void sortByCname() throws Exception{
            ps = con.prepareStatement("SELECT * FROM CRICKETER ORDER BY CNAME");
            rs = ps.executeQuery();
            while(rs.next()){
              System.out.printf("%-15s%-20s%5s\n",rs.getString(1),rs.getString(2),rs.getString(3));
            }
    }

    public void sortByScore() throws Exception{
            ps = con.prepareStatement("SELECT * FROM CRICKETER ORDER BY SCORE DESC");
            rs = ps.executeQuery();
            while(rs.next()){
              System.out.printf("%-15s%-20s%5s\n",rs.getString(1),rs.getString(2),rs.getString(3));
            }
    }

    public void searchByCcode(int ccode) throws Exception{
            ps = con.prepareStatement("SELECT * FROM CRICKETER WHERE CCODE = ?");
            ps.setInt(1,ccode);
            rs = ps.executeQuery();
            if(rs.next())
             System.out.printf("%-15s%-20s%5s\n",rs.getString(1),rs.getString(2),rs.getString(3));
            else
                    System.out.println("Record Not Found ...!");
    }

    public void searchByScore(int score) throws Exception{
            boolean found = false;
            ps = con.prepareStatement("SELECT * FROM CRICKETER WHERE SCORE > ?");
            ps.setInt(1,score);
            rs = ps.executeQuery();
            while(rs.next()){
              System.out.printf("%-15s%-20s%5s\n",rs.getString(1),rs.getString(2),rs.getString(3));
              found=true;
            }
            if(!found)
                    System.out.println("Record Not Found ...!");
    }

    public void close() throws Exception{
            con.close();
    }
}
```

```
import java.io.*;

class CricketerCRUDDemo{
	public static void main(String[] args) throws Exception {
		int choice = -1;
		BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
		DBConnection con = new DBConnection();

		do{
			System.out.println("1. INSERT");
			System.out.println("2. DISPLAY ALL");
			System.out.println("3. SEARCH BY CCODE");
			System.out.println("4. SEARCH BY SCORE");
			System.out.println("5. UPDATE");
			System.out.println("6. DELETE");
			System.out.println("7. SORT BY SCORE");
			System.out.println("8. SORT BY CNAME");
			System.out.println("0. EXIT");
			System.out.print("Enter Your Choice : ");
			choice = Integer.parseInt(br.readLine());

			switch(choice){
				case 1:
					System.out.print("Enter how many cricketer you want : ");
					int n = Integer.parseInt(br.readLine());

					for(int i=0;i<n;i++){
						System.out.print("Enter Cricketer Code : ");
						int ccode = Integer.parseInt(br.readLine());

						System.out.print("Enter Cricketer Name : ");
						String cname = br.readLine();

						System.out.print("Enter Cricketer Score : ");
						int score = Integer.parseInt(br.readLine());

						con.insert(ccode,cname,score);
					}
				break;
				case 2:
					con.display();
				break;
				case 3:
						System.out.print("Enter Cricketer Code to SEARCH : ");
						int ccode = Integer.parseInt(br.readLine());
						con.searchByCcode(ccode);
				break;
				case 4:
						System.out.print("Enter Cricketer Score to Search : ");
```

```
                                int score = Integer.parseInt(br.readLine());
                                con.searchByScore(score);
                    break;
                    case 5:

                                System.out.print("Enter Cricketer Code to Update : ");
                                ccode = Integer.parseInt(br.readLine());

                                System.out.print("Enter New Name: ");
                                String cname = br.readLine();

                                System.out.print("Enter New Score : ");
                                score = Integer.parseInt(br.readLine());
                                con.update(ccode,cname,score);

                    break;

                    case 6:
                                System.out.print("Enter Cricketer Code to Delete : ");
                                ccode = Integer.parseInt(br.readLine());
                                con.delete(ccode);
                    break;
                    case 7:
                           con.sortByScore();
                    break;
                    case 8:
                           con.sortByCname();
                    break;
                    case 0:

                                con.close();
                                System.out.println("Thanks...Bye... !");
                    break;
               }
          }while(choice!=0);

     }
}
```