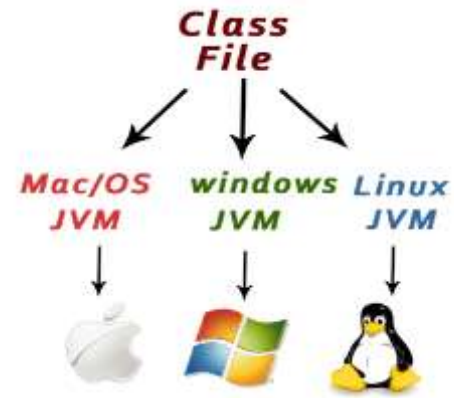
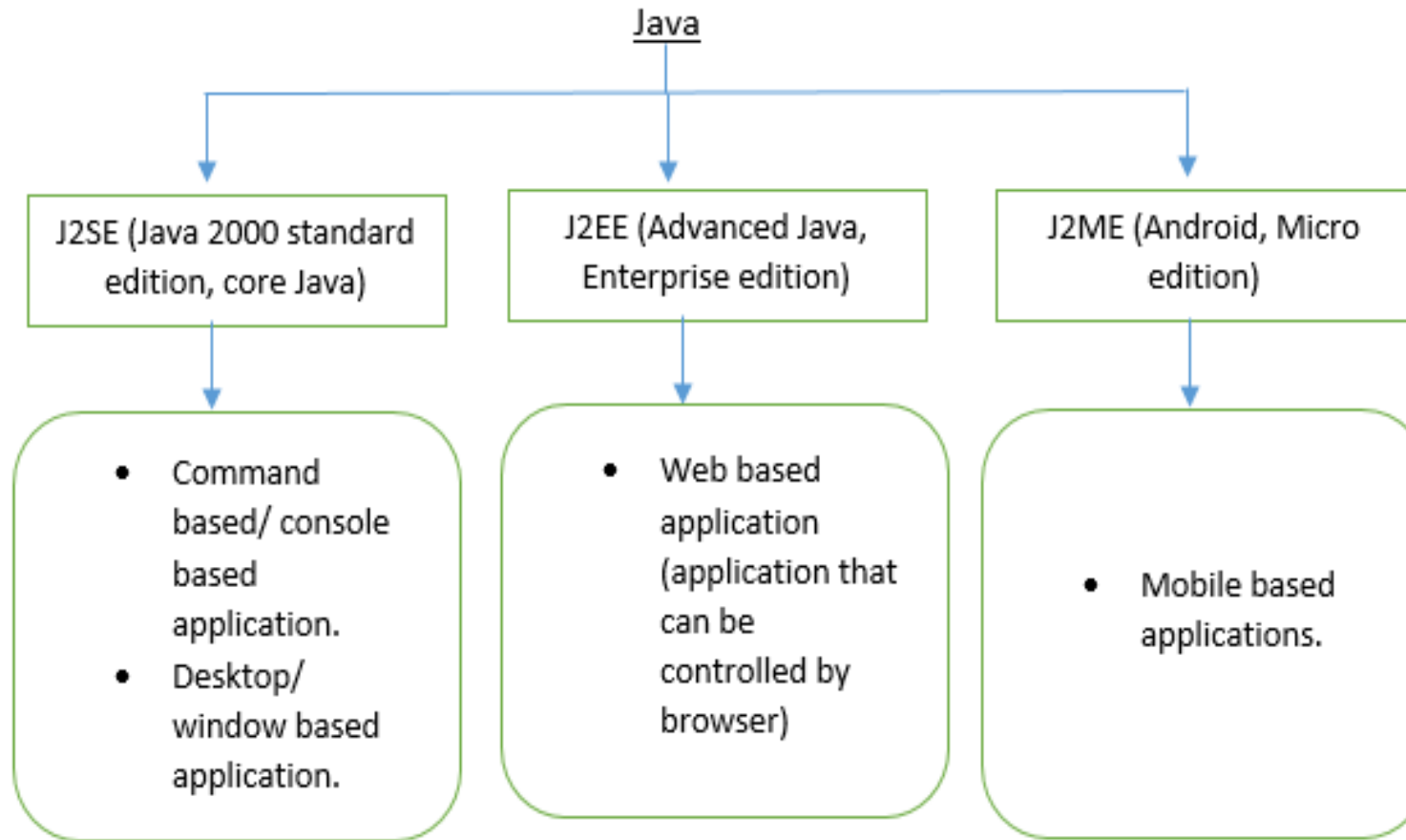


WHY JAVA

- It's almost entirely **Object-Oriented Programming**
- It's more **Platform Independent**
- It's more **Secure** than the **other language**
- Provides many generic services like **scalability**,
- **transaction management, life cycle management** etc



JAVA EDITIONS

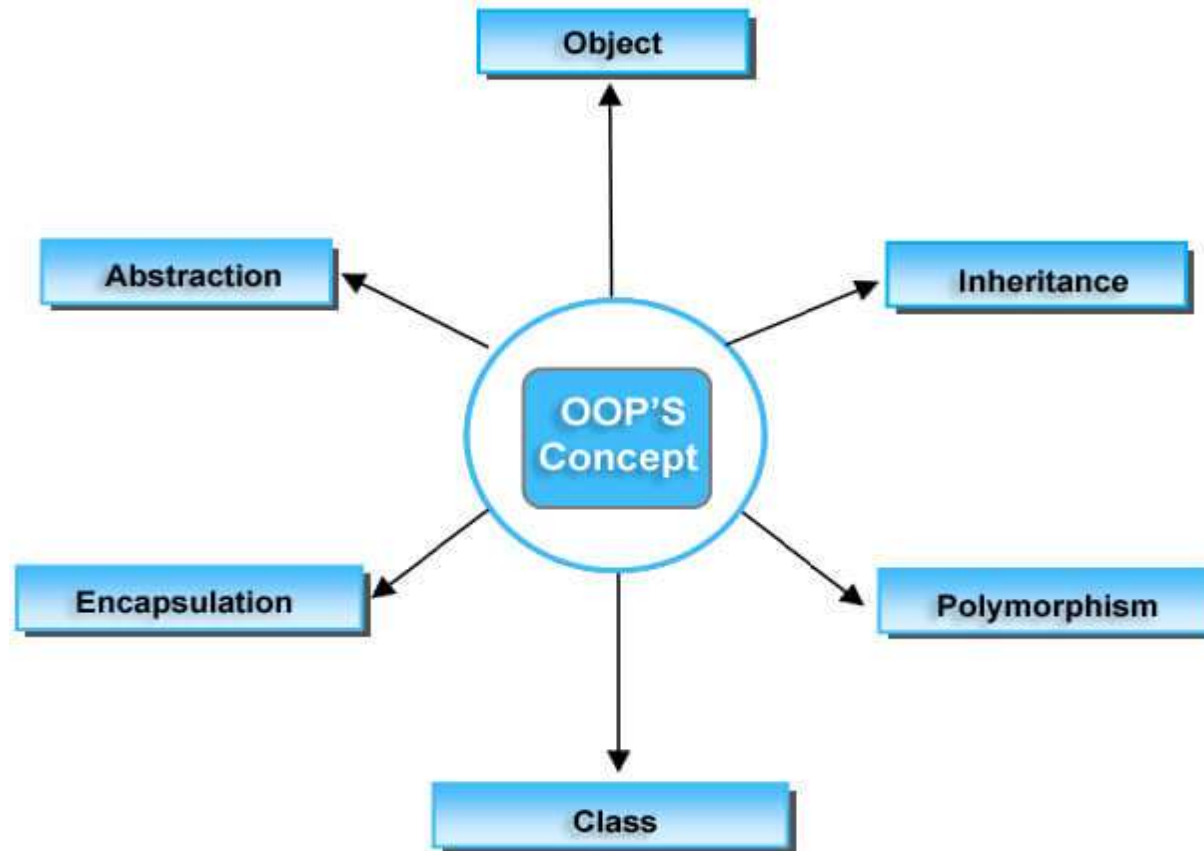


Programming languages used in most popular websites*

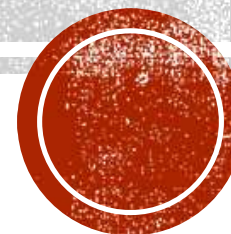
Websites	Front-end	Back-end	Database	Notes
	(Client-side)	(Server-side)		
Google.com	JavaScript	C, C++, Go, Java, Python, PHP (HHVM)	Bigtable, MariaDB	The most used search engine in the world
Facebook.com	JavaScript	Hack, PHP (HHVM), Python, C++, Java, Erlang, D, XHP, Haskell	MariaDB, MySQL, HBase, Cassandra	The most visited social networking site
YouTube.com	JavaScript	C, C++, Python, Java, Go	Vitess, BigTable, MariaDB	The most visited video sharing site
Yahoo	JavaScript	PHP	PostgreSQL, HBase, Cassandra, MongoDB	Yahoo is presently transitioning to Node.js
Amazon.com	JavaScript	Java, C++, Perl	Oracle Database	Popular internet shopping site
Wikipedia.org	JavaScript	PHP, Hack	MariaDB	programmed in PHP, runs on HHVM; free
Twitter.com	JavaScript	C++, Java, Scala, Ruby	MySQL	Popular social network.
eBay.com	JavaScript	Java, JavaScript ¹ Scala	Oracle Database	Online auction house
Microsoft	JavaScript	C#	Microsoft SQL Server	largest software companies.
Linkedin.com	JavaScript	Java, JavaScript, Scala	Voldemort	World's largest professional network.
Pinterest	JavaScript	Python (Django), Erlang	MySQL, Redis	
WordPress.com	JavaScript	PHP	PostgreSQL, HBase, Cassandra, MongoDB,	



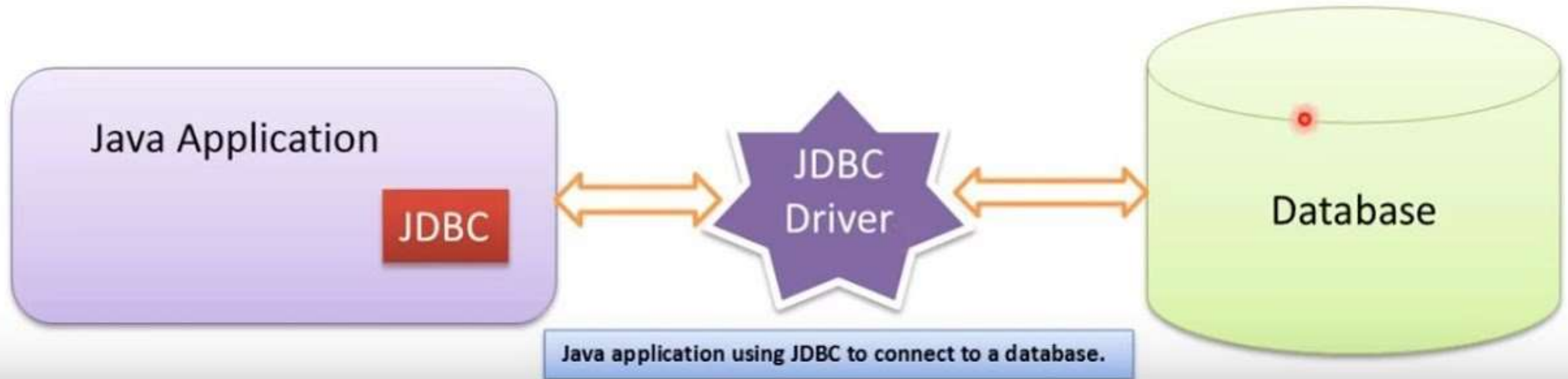
OOP'S CONCEPT



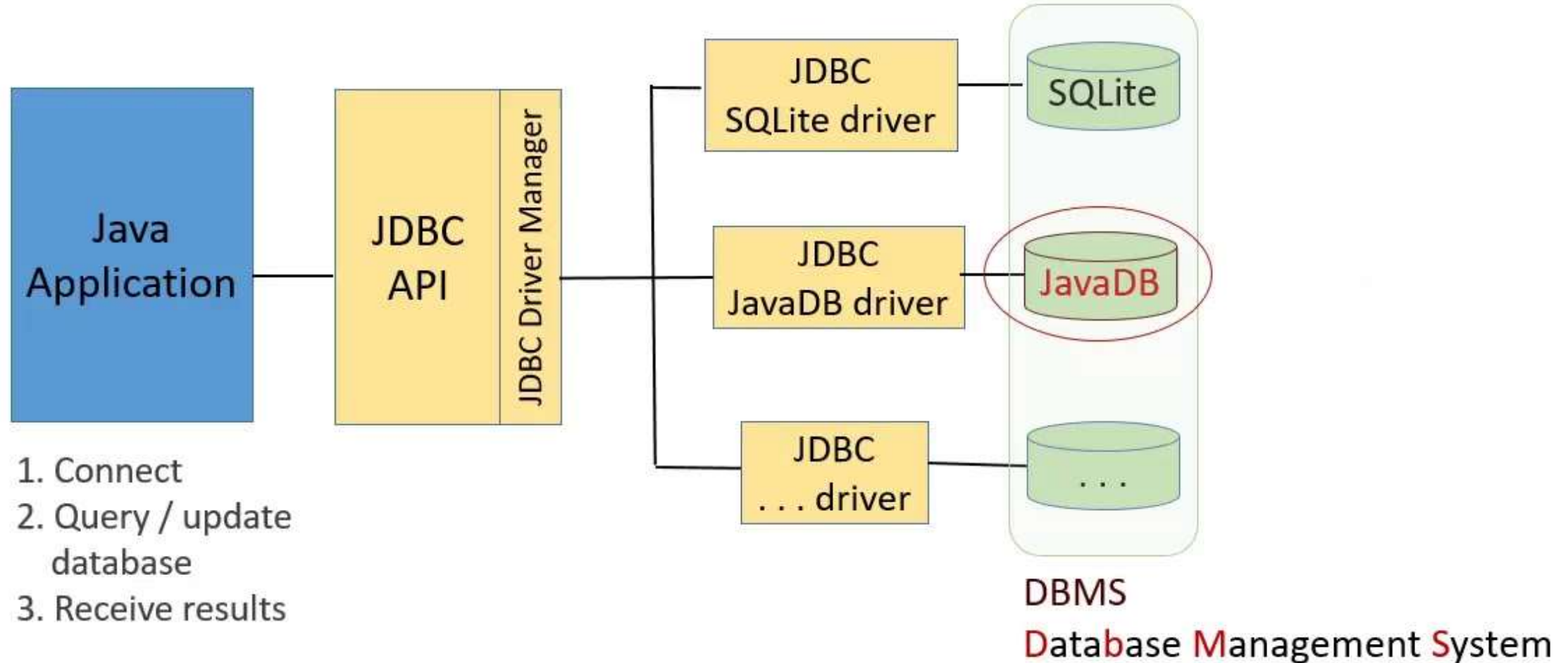
JDBC



JDBC Introduction



JDBC - Java Database Connectivity



WHY USE JDBC?

- Before JDBC, ODBC API was used to connect and execute query to the database
- But ODBC API uses ODBC driver that is written in C language which is platform dependent and unsecured
- Sun Microsystem has defined its own API (JDBC API) that uses JDBC driver written in Java language.



JDBC - ODBC Bridge
Driver

Network Protocol
Driver

Native Driver



Thin Driver

JDBC in Java



JDBC DRIVER

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

There are 4 types of JDBC drivers:

- JDBC-ODBC bridge driver
- Native-API driver (partially java driver)
- Network Protocol driver (fully java driver)
- Thin driver (fully java driver)



JDBC-ODBC BRIDGE DRIVER

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.



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.



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.



THIN DRIVER

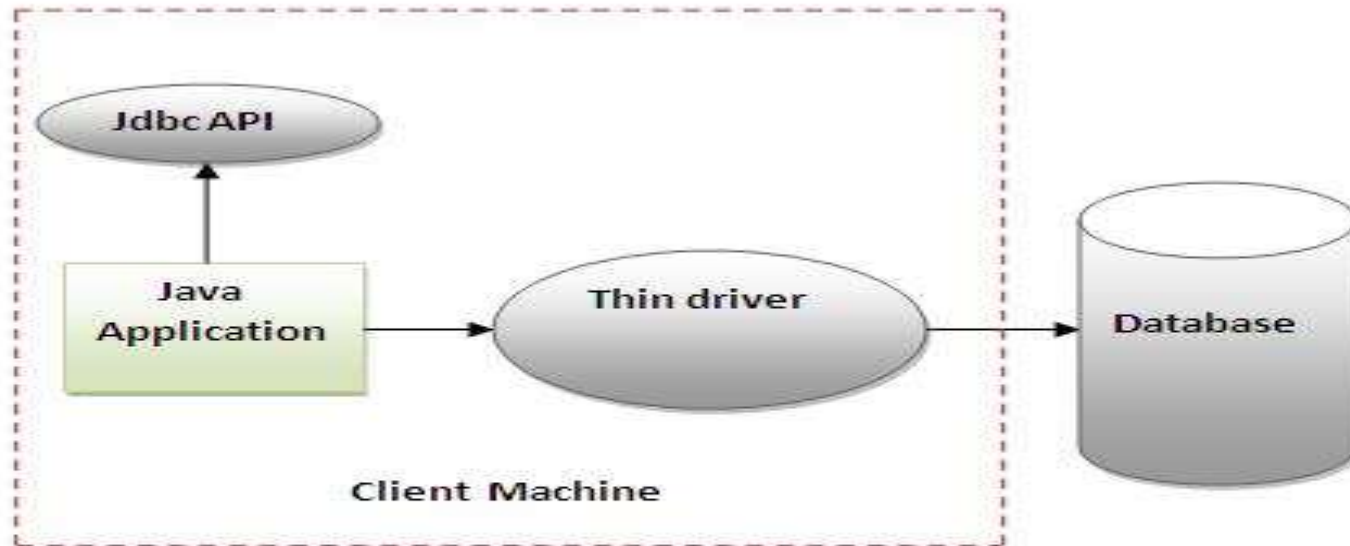


Figure- Thin Driver



THIN DRIVER

Advantage:

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

Disadvantage:

- Drivers depends on the Database.

- **Mysql**

```
Class.forName("com.mysql.jdbc.Driver");
```

```
DriverManager.getConnection("jdbc:mysql://localhost:3306/jdbcdb", "root", "Jaibrakash@7");
```

- **Oracle**

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

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



5 STEPS TO CONNECT TO THE DATABASE IN JAVA

- There are 5 steps to connect any java application with the database in java using JDBC.

They are as follows:

- Register the driver class
- Creating connection
- Creating statement
- Executing queries
- Closing connection



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.



CREATE THE CONNECTION OBJECT

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



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



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 to execute query

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



CLOSE THE CONNECTION OBJECT

- By closing connection object statement and ResultSet will be closed automatically.

Example to close connection

- `con.close();`



CREATE STATEMENT USING CREATE TABLE IN DATABASE

```
import java.sql.*;
import java.io.*;
class createstable
{
    public static void main(String args[])
    {
        try
        {
            Class.forName("com.mysql.jdbc.Driver");
            Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/jdbcdb", "root",
"Jaibrakash@7");
            String qry = "create table flower( fid int, fname varchar(50), fdesc varchar(50), fprice varchar(50)";
                Statement s = con.createStatement();
                s.execute(qry);
                System.out.println("Table created");
            s.close();
            con.close();
        }catch(Exception e){}
        System.out.println("Successfully Created");
    }
}
```



CREATE STATEMENT USING INSERT INTO DATABASE

```
import java.sql.*;
import java.io.*;
class createInsert
{
    public static void main(String args[])
    {    try
        { Class.forName("com.mysql.jdbc.Driver");
Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/jdbcdb",
"root","Jaibrakash@7");
        Statement st=con.createStatement();
        st.executeUpdate("insert into flower values(101,'Lotus','Flower',200)");
        st.close();
        con.close();
    }catch(Exception e){}
    System.out.println("Record Successfully inserted");
```



CREATE STATEMENT USING UPDATE INTO DATABASE

```
import java.sql.*;
import java.io.*;
class createupdate
{public static void main(String args[])
{try
{Class.forName("com.mysql.jdbc.Driver");
Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/jdbcdb",
"root","Jaibrakash@7");
String qry = "update flower set fname="Lotus" where fid=101";
        Statement s = con.createStatement();
        s.executeUpdate(qry);
        System.out.println(" Updated Successfully into the database");
        s.close();
        con.close();
    }
    catch(Exception e){}}
```



CREATE STATEMENT USING DELETE FROM DATABASE

```
import java.sql.*;
import java.io.*;
class createupdate
{
public static void main(String args[])
{
try
{
Class.forName("com.mysql.jdbc.Driver");
Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/jdbcdcb",
"root","Jaibrakash@7");

String qry = "delete from flower where fid=101";
Statement s = con.createStatement( );
s.executeUpdate(qry );
System.out.println(" Delete Successfully from the database");
s.close();
con.close();
}
catch(Exception e){} }}
```



PREPAREDSTATEMENT INTERFACE

- The PreparedStatement interface is a subinterface of Statement. It is used to execute parameterized query.
- parameterized query:
- `String sql="insert into emp values(?,?,?)";`



WHY USE PREPAREDSTATEMENT?

Improves performance

- The performance of the application will be faster.
- PreparedStatement interface because query is compiled only once.



METHODS OF PREPAREDSTATEMENT INTERFACE

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.



PREPARESTATEMENT USING CREATE TABLE IN DATABASE

```
import java.sql.*;
import java.io.*;
class preparestcreate
{
public static void main(String args[])
{   try {
Class.forName("com.mysql.jdbc.Driver");
Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/jdbcdb",
    "root","Jaibrakash@7");
String qry = "create table empl ( emp_id int, emp_name varchar(50), emp_addr varchar(50)";
PreparedStatement stmt=con.prepareStatement(qry);
    stmt.execute();
        System.out.println("Table created");
            stmt.close();
                con.close();
    }
catch(Exception e){}
}
```



```
package com.centum;

import java.sql.*;

public class PrepareCreate {

    public static void main(String args[]) {
        try {
            Class.forName("com.mysql.jdbc.Driver");
            Connection con = DriverManager.getConnection(
                "jdbc:mysql://localhost:3306/centumdb", "root",
                "Jaibrakash@7");

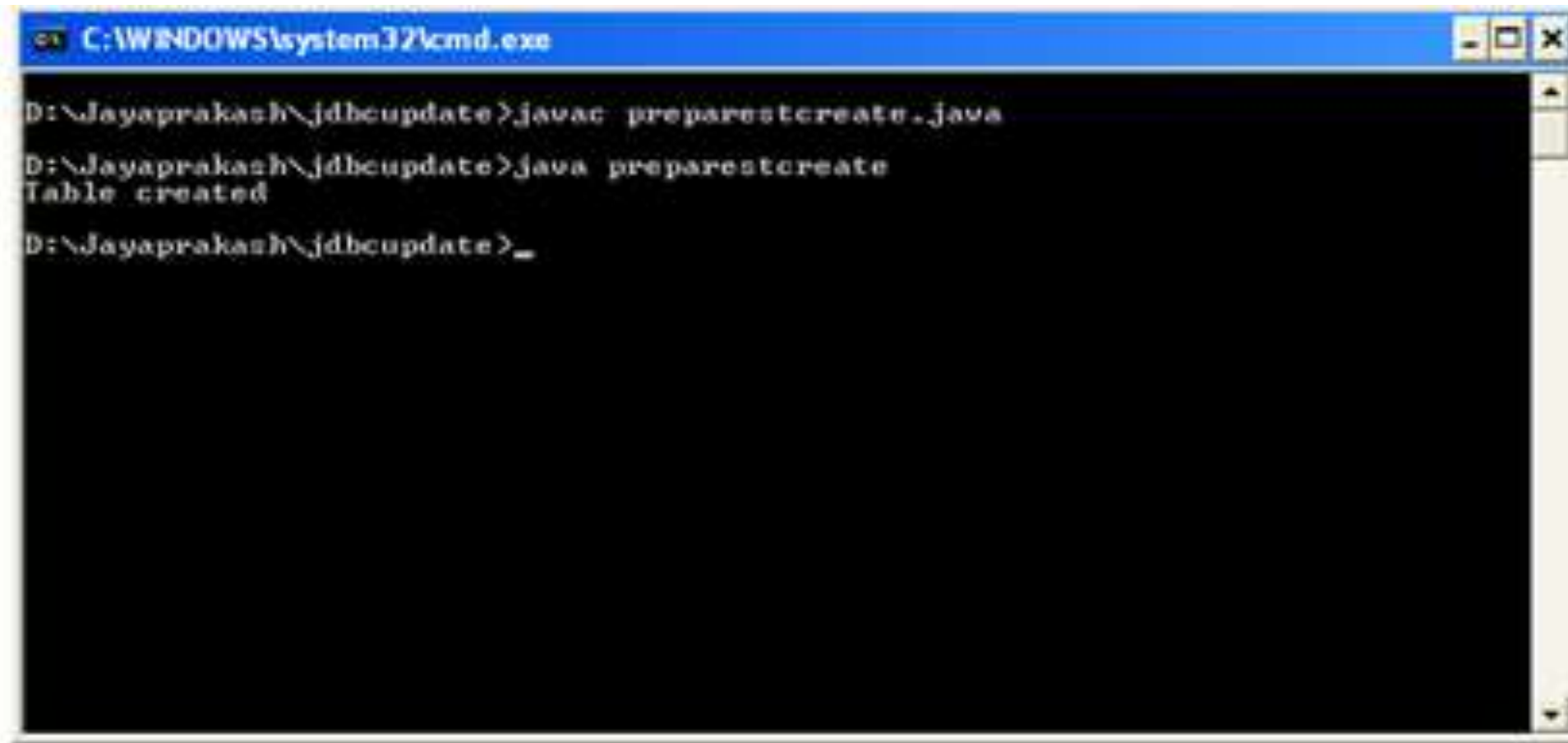
            String qry = "create table student123( sid int not null AUTO_INCREMENT, sname varchar(50) NOT NULL, "
                + "saddress varchar(50) NOT NULL, city varchar(50) NOT NULL, primary key(sid))";
            PreparedStatement stmt = con.prepareStatement(qry);
            stmt.execute();

            stmt.close();
            con.close();
            System.out.println("Table created");

        } catch (SQLException | ClassNotFoundException e) {
            e.printStackTrace();
        }
        System.out.println("Successfully Created ..");
    }
}
```



OUTPUT:



```
C:\WINDOWS\system32\cmd.exe

D:\Jayaprakash\jdbcupdate>javac preparestcreate.java

D:\Jayaprakash\jdbcupdate>java preparestcreate
Table created

D:\Jayaprakash\jdbcupdate>_
```



PREPARESTATEMENT USING INSERT INTO DATABASE

```
import java.sql.*;
```

```
import java.io.*;
```

```
class preparestinse
```

```
{ public static void main(String args[])
```

```
{ try {
```

```
Class.forName("com.mysql.jdbc.Driver");
```

```
Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/jdbcdb",  
"root","Jaibrakash@7");
```

```
String qry = "insert into empl values ( ?, ?, ?)";
```

```
    PreparedStatement ps = con.prepareStatement( qry );
```

```
    ps.setInt(1,444);
```

```
    ps.setString(2,"Poorna");
```

```
    ps.setString(3,"AnnaNagar");
```

```
        int i=ps.executeUpdate();
```

```
        System.out.println("No of rows inserted = " + i);
```

```
    ps.close();
```

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

```
    } catch (Exception e){
```



PREPARE STATEMENT USING UPDATE INTO DATABASE

```
import java.sql.*;
import java.io.*;
class preparestUpd
{public static void main(String args[])
{ try {
Class.forName("com.mysql.jdbc.Driver");
Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/jdbcdb",
"root","Jaibrakash@7");
PreparedStatement stmt=con.prepareStatement("update empl set emp_name=? where emp_id=?");
stmt.setString(1,"Hema");//1 specifies the first parameter in the query i.e. emp_name
stmt.setInt(2,20); //1 specifies the second parameter in the query i.e. emp_id
int i=stmt.executeUpdate();
System.out.println(i+" records updated");
con.close();} catch(Exception e){} } }
```



PREPARESTATEMENT USING DELETE FROM DATABASE

```
import java.sql.*;
import java.io.*;
class preparestdel
{
public static void main(String args[])
{ try {
Class.forName("com.mysql.jdbc.Driver");
Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/jdbcdcb", "root","Jaibrakash@7");
PreparedStatement stmt=con.prepareStatement("delete from empl where emp_id=?");
stmt.setInt(1,20);
int i=stmt.executeUpdate();
System.out.println(i+" records deleted");
con.close();
}
catch(Exception e){}
}
}
```





RESULTSETMETADATA INTERFACE

- The metadata means data about data
- i.e. we can get further information from the data.
- to get metadata of a table like total number of column, column name, column type etc



COMMONLY USED METHODS OF RESULTSETMETADATA INTERFACE

Method	Description
public int getColumnCount()throws SQLException	it returns the total number of columns in the ResultSet object.
public String getColumnName(int index)throws SQLException	it returns the column name of the specified column index.
public String getColumnName(int index)throws SQLException	it returns the column type name for the specified index.
public String getTableName(int index)throws SQLException	it returns the table name for the specified column index.



EXAMPLE

```
import java.sql.*;
class Rsmd{
public static void main(String args[]){
try{
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection( "jdbc:oracle:thin:@localhost:1521:xe","system"
,"oracle");

PreparedStatement ps=con.prepareStatement("select * from emp1");
ResultSet rs=ps.executeQuery();
ResultSetMetaData rsmd=rs.getMetaData();

System.out.println("Total columns: "+rsmd.getColumnCount());
System.out.println("Column Name of 1st column: "+rsmd.getColumnName(1));
System.out.println("Column Type Name of 1st column: "+rsmd.getColumnTypeName(1));
con.close();

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



OUTPUT:

Total columns: 2

Column Name of 1st column: ID

Column Type Name of 1st column: NUMBER



DATABASEMETADATA INTERFACE:

- **DatabaseMetaData interface** provides methods to get meta data of a database
- Database **product name, database product version, driver name,**
name of total number of tables, name of total number of views etc



COMMONLY USED METHODS OF DATABASEMETADATA INTERFACE

- **public String getDriverName()** throws **SQLException**: it returns the name of the JDBC driver.
- **public String getDriverVersion()** throws **SQLException**: it returns the version number of the JDBC driver.
- **public String getUsername()** throws **SQLException**: it returns the username of the database.
- **public String getDatabaseProductName()** throws **SQLException**: it returns the product name of the database.
- **public String getDatabaseProductVersion()** throws **SQLException**: it returns the product version of the database



SIMPLE EXAMPLE OF DATABASEMETADATAINTERFACE :

```
import java.sql.*;
class Dbmd{
public static void main(String args[]){
try{
    Class.forName("oracle.jdbc.driver.OracleDriver");
    Connection con=DriverManager.getConnection( "jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
    DatabaseMetaData dbmd=con.getMetaData();
    System.out.println("Driver Name: "+dbmd.getDriverName());
    System.out.println("Driver Version: "+dbmd.getDriverVersion());
    System.out.println("Database Product Name: "+dbmd.getDatabaseProductName());
    System.out.println("Database Product Version: "+dbmd.getDatabaseProductVersion());

    con.close();

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



OUTPUT:

Driver Name: Oracle JDBC Driver

Driver Version: 10.2.0.1.0XE

Database Product Name: Oracle

Database Product Version: Oracle Database 10g Express Edition
Release 10.2.0.1.0 -Production

