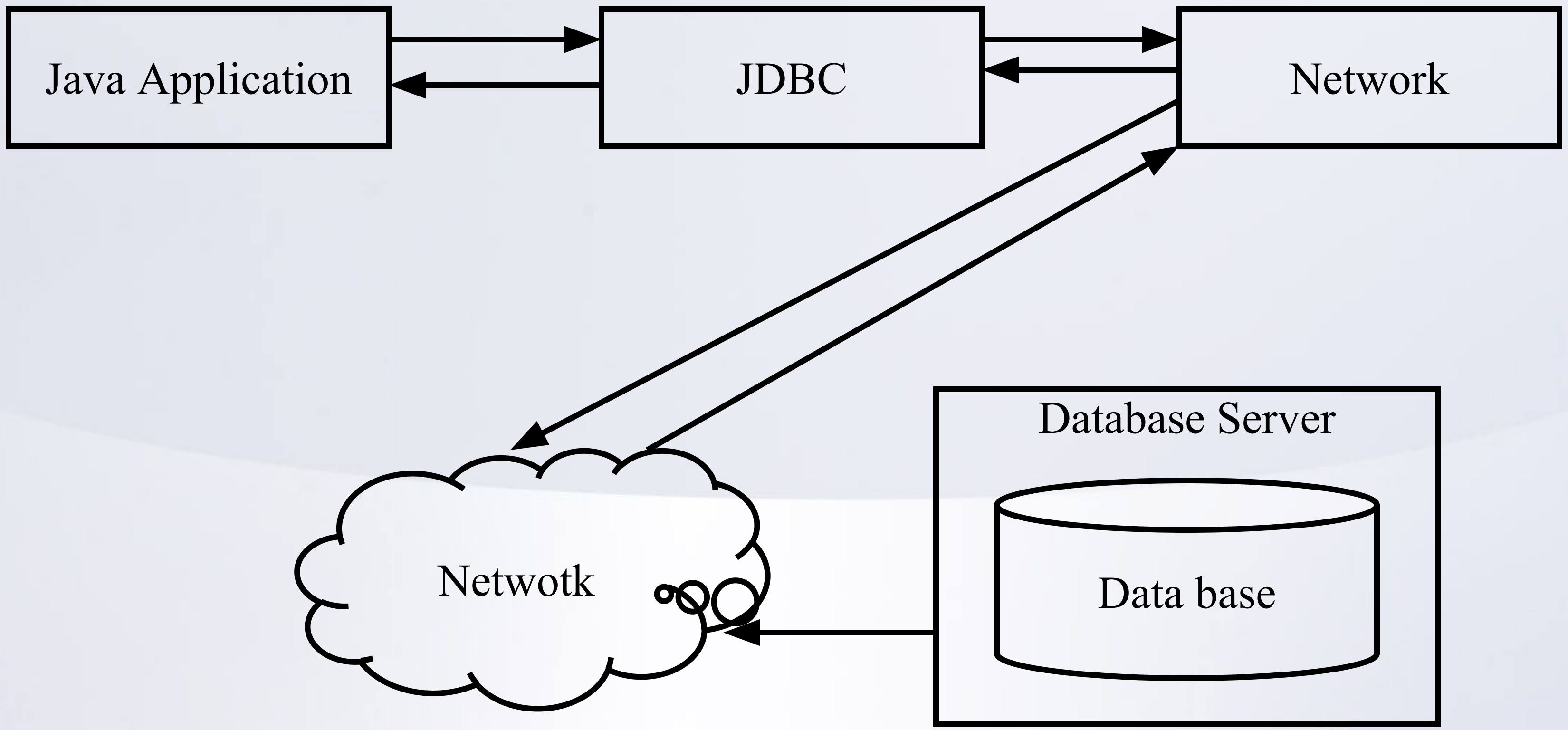
Lets Start With JDBC……………

JDBC vs. ODBC

JDBC	ODBC
JDBC can directly used with Java because it uses **Java Interface**.	ODBC can not be directly used with Java because it **uses a C interface**.
There is **no Native code**, so **drawbacks like security, implementation are not there.**	Calls from Java to **native C code** have number of **drawbacks in the security, implementation, robustness and automatic portability** of applications.
JDBC do **not use pointers** because it is written in JAVA.	ODBC **makes use of pointers** which have been totally removed from Java.
JDBC is designed to keep things **simple** while allowing advance capabilities when required.	ODBC mixes simple advanced features together and has **complex** options for simple queries.
JDBC drivers are written in Java and JDBC code is **automatically installable**, secure and **portable** on all Java platforms.	**ODBC requires manual installation** of the ODBC driver manager and driver on all client machines.

Functionality of JDBC



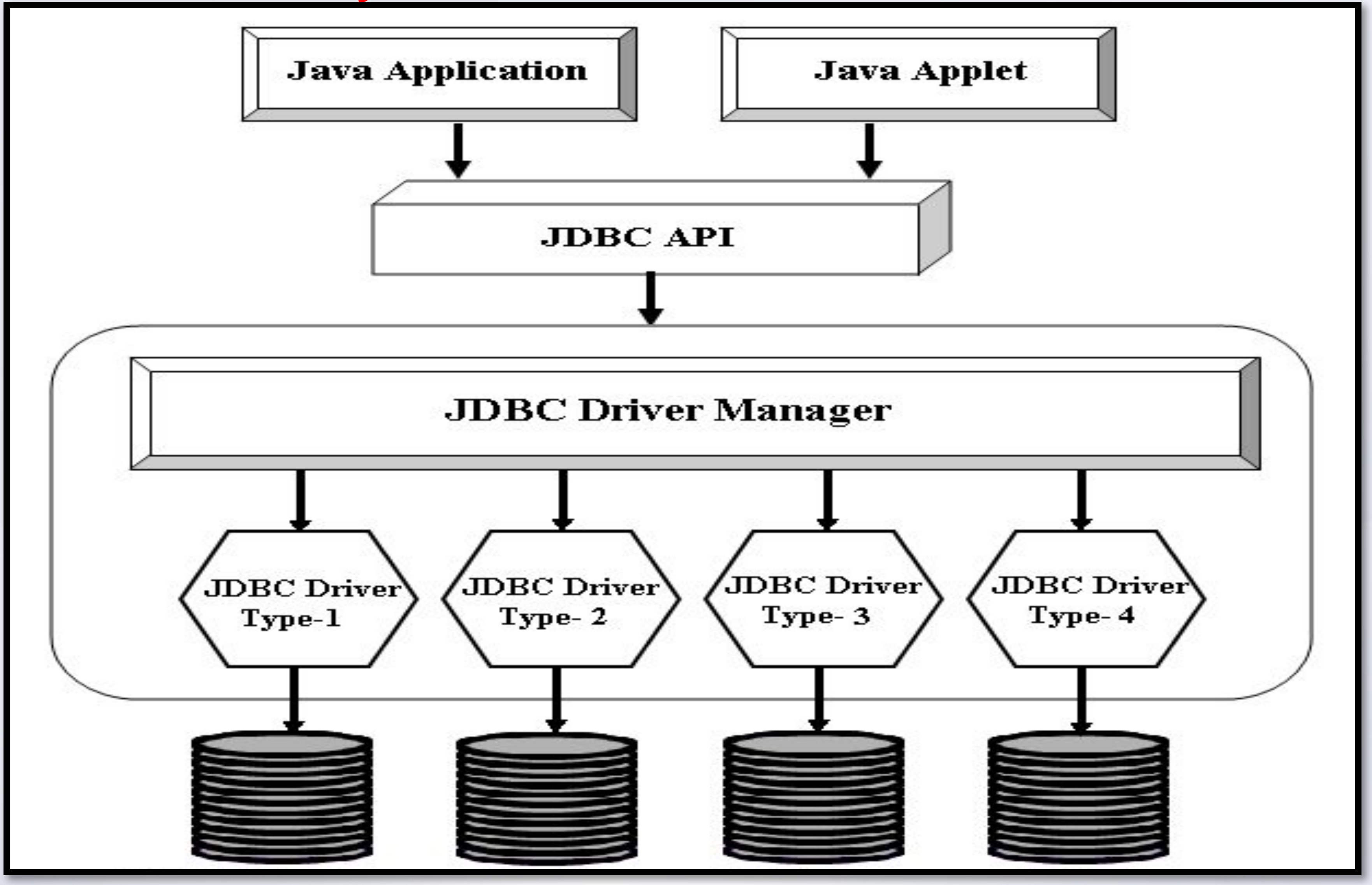
JDBC |Java Database Connectivity

- Java runs on a Java Virtual Machine (JVM). JVM translates your application code to byte codes.
- A database is a separate software system. In order for these to talk, they need to have a communication channel.
- Sun offers, through its J2SE, a package entirely dedicated for performing database-related operations. This is the JDBC API.

Java Database Connectivity

- **Advantages :**
  - Supports variety of relation databases –Provide existing enterprise data
  - Easy to develop enterprise application
  - Zero configuration for network computer – No client side installation
  - Short development time
  - Doesn’t require special installation

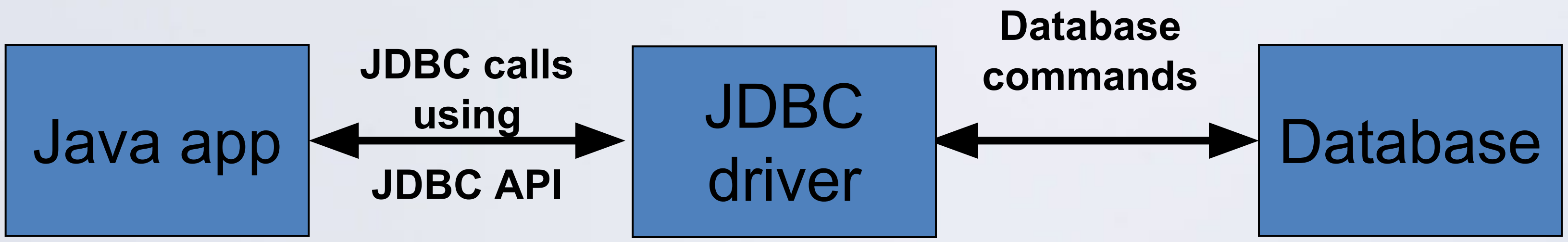
JDBC Architecture



Concept of Driver

- Just the same way, your printer needs a driver, or maybe, your new soundcard needs a driver for the underlying hardware to understand, a database needs a driver. A database can be accessed by proprietary API's.
- The Java programmer makes JDBC calls from his program.
- The JVM translates the code to the database API.
- One needs a database driver either from a database vendor, or a J2EE server vendor.

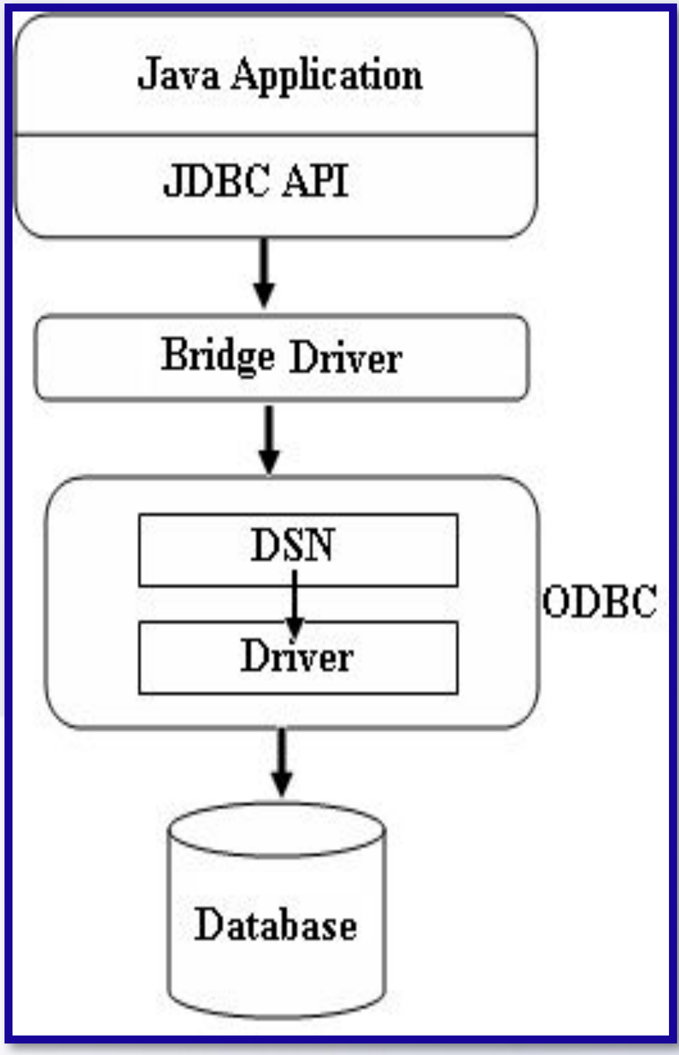
JDBC |Java Database Connectivity



JDBC Driver Types

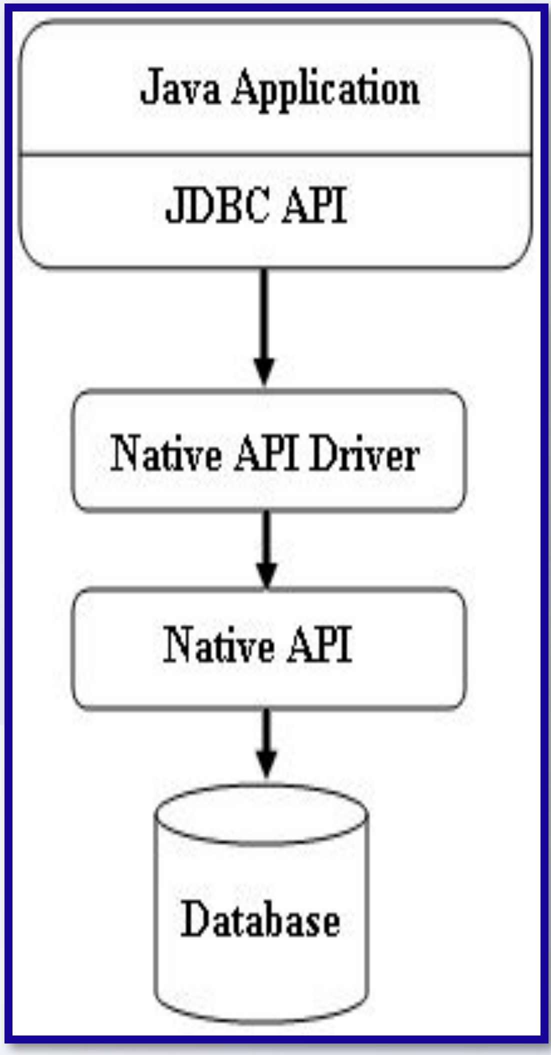
Type 1: JDBC-ODBC Bridge

- **Advantages:**
  - allows access to almost any database
- **Disadvantages:**
  - Not portable.
  - Performance is slowest in compare to all drivers….
  - The client system requires the ODBC Installation to use the driver.
  - Not good for the Web Application or for large scale database based applications.



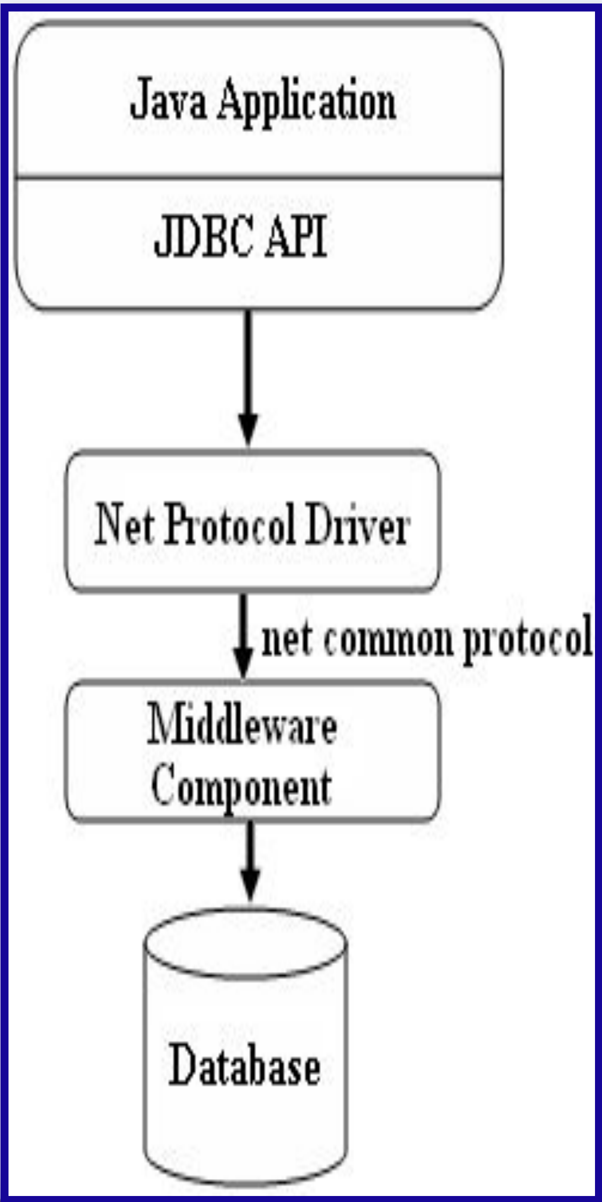
Type 2: Native-API / partly-Java driver

- Advantages:
  - offer better performance than the Type 1 as the layers of communication (tiers) are less than and it uses Native API which is Database specific.
- Disadvantages:
  - Native API must be installed in the Client System and hence type 2 drivers cannot be used for the Internet.
  - portability issue.
  - If we change the Database we have to change the Native API as it is specific to a database.
  - Mostly outdated now because it is not thread safe.
  - It is not suitable for distributed application.



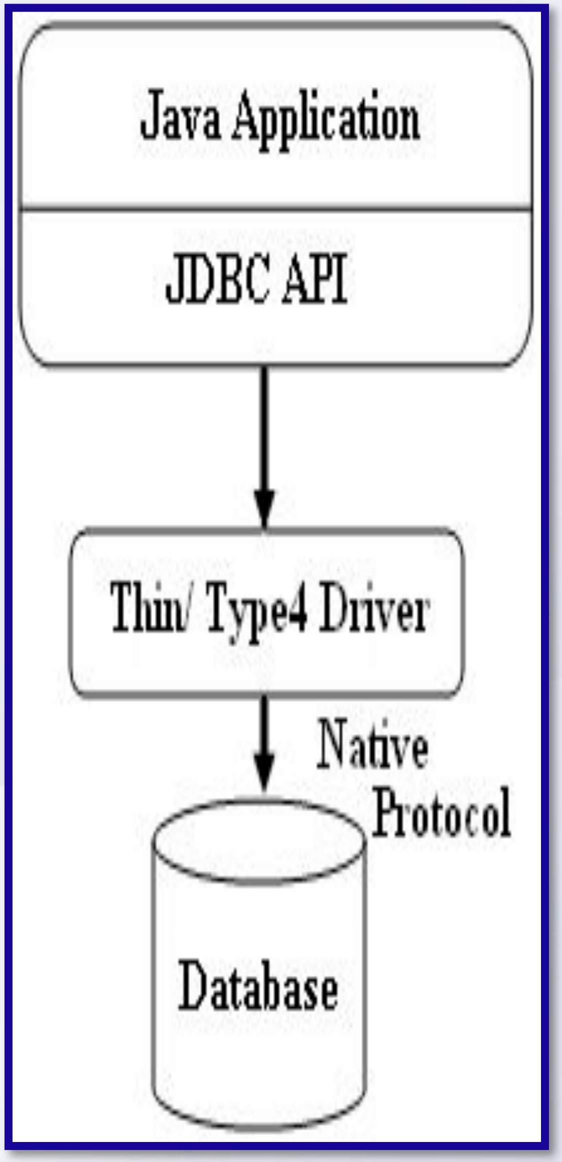
Type 3: Net-protocol/all-Java Driver

- Advantages:
  - server-based: no need for any vendor DB library to be present on client.
  - Portable.
  - designed to make the client driver very small and fast to load.
  - provides support for features such as caching (connections, query results, and so on), load balancing, and advanced system administration.
  - very flexible allows access to multiple databases using one driver.
  - most efficient amongst all driver types.
- Disadvantages:
  - Requires database-specific coding to be done in the middle tier and it requires additional server for that.



Type 4: Native-protocol/ all-Java Driver

- Advantages:
  - Portable and Platform independent and using this benefit we can reduce deployment administration issues. So, it is most suitable for the java based web application.
  - Number of translation layers is very less. So, performance is typically quite good.
  - You need not to install special software on the client or server. Further, these drivers can be downloaded dynamically.
- Disadvantages:
  - A different driver is needed for each and every database.



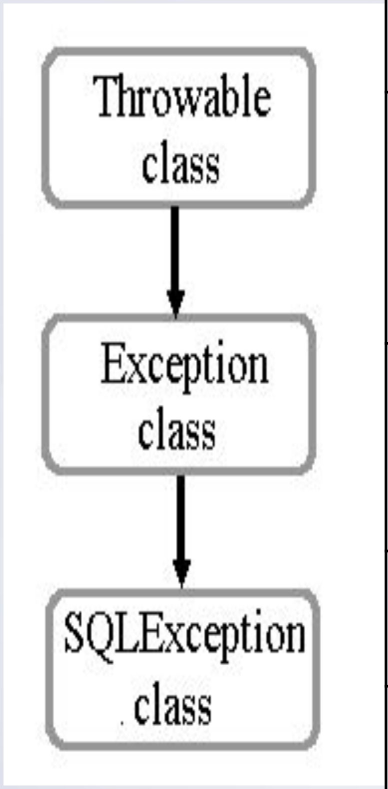
Overview of JDBC API

The JDBC API is available in **the java.sql and javax.sql** packages.

1. **DriverManager Class-** Loads JDBC drivers in memory. Can also be used to open connections to a data source.
2. **Connection Interface-** Represents a connection with a data source. Is also used for creating Statement, PreparedStatement and CallableStatement objects.
3. **Statement Interface-** Represents a static SQL statement. Can be used to retrieve ResultSet object/s.
4. **PreparedStatement Interface** - Higher performance alternative to Statement object, represents a precompiled SQL statement.
5. **CallableStatement Interface** - Represents a stored procedure. Can be used to execute stored procedures in a RDBMS which supports them.
6. **ResultSet Interface-** Represents a database result set generated by using a SELECT SQL statement.
7. **ResultSetMetaData Interface** – Represents information about result set object.
8. **DatabaseMetaData Interface** - Represents information about database.
9. **SQLException** - An exception class which encapsulates database base access errors.

SQLException Class

Hierarchy of SQLException class and Its Methods



Return Type	Method	Description
int	getErrorCode()	It is used to retrieve the vendor-specific exce ption code for this SQLException object
SQLExceptio n	getNextException()	It is used to retrive the exception chained to this SQLException object.
String	getSQLState()	Retrieves the SQLState for this SQLException object
void	setNextException (SQLException ex)	Adds a SQLException object to the end of the chain

JDBC steps

1. Connect to database
   1. Load the driver
   2. Define the Connection URL
   3. Establish the Connection
2. Query database (or insert/update/delete)
   1. Create a Statement object
   2. Execute a query
3. Process results
4. Close connection to database

Step 1 Connect to Database

i. Loading the driver

public static Class **forName** (String className) throws **ClassNotFoundException**

Example:

```
try
 {
          Class.forName("sun.jdbc.odbc.JdbcOdbcDriver"); or
          Class.forName(“com.mysql.jdbc.Driver”); //mysql

  }
  catch(ClassNotFoundException e) { }
```

Step 1….

ii. Define the Connection URL

Example:

String url="jdbc:odbc:test“ (test is DSN)

or

String url="jdbc:odbc:Driver={Microsoft Access Driver(*.mdb)}; DBQ=d:\\stud.mdb”

or

String url=“"jdbc:mysql://localhost:3306/<dbname>",”<usernm>",”<password>"

Step 1....

iii. Establish the Connection

public static Connection **getConnection** (String url, String user, String password) throws SQLException

Example for Oracle:

Connection con= DriverManager.getConnection(url, “system”,”manager”);

Example for Ms Access:

Connection con= DriverManager.getConnection(url, “ ”,” ”);

Step 2 Query to Database

Create Statement Object for Execute Query and ResultSet object to store the result.

Ex.

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

ResultSet rs = stmt.executeQuery("Select * from info");
```

Connection Interface

- ✔ Connection interface defines connection to the different databases.
- ✔ An instance of the connection interface obtained from the getConnection () method of DriverManager class.
- ✔ It is also able to get the information about table structure of database, its supported SQL grammar, its stored procedures, the capabilities of this connection, and so on.

Connection Interface Methods

1. Statement createStatement()
2. PreparedStatement prepareStatement (String sql)
3. CallableStatement prepareCall(String sql)
4. void close()
5. void commit()
6. void rollback()
7. void setAutoCommit(boolean autoCommit)
8. boolean getAutoCommit()
9. boolean isClosed()
10. DatabaseMetaData getMetaData()

DriverManager Class

The DriverManager class is available in the java.sql package.

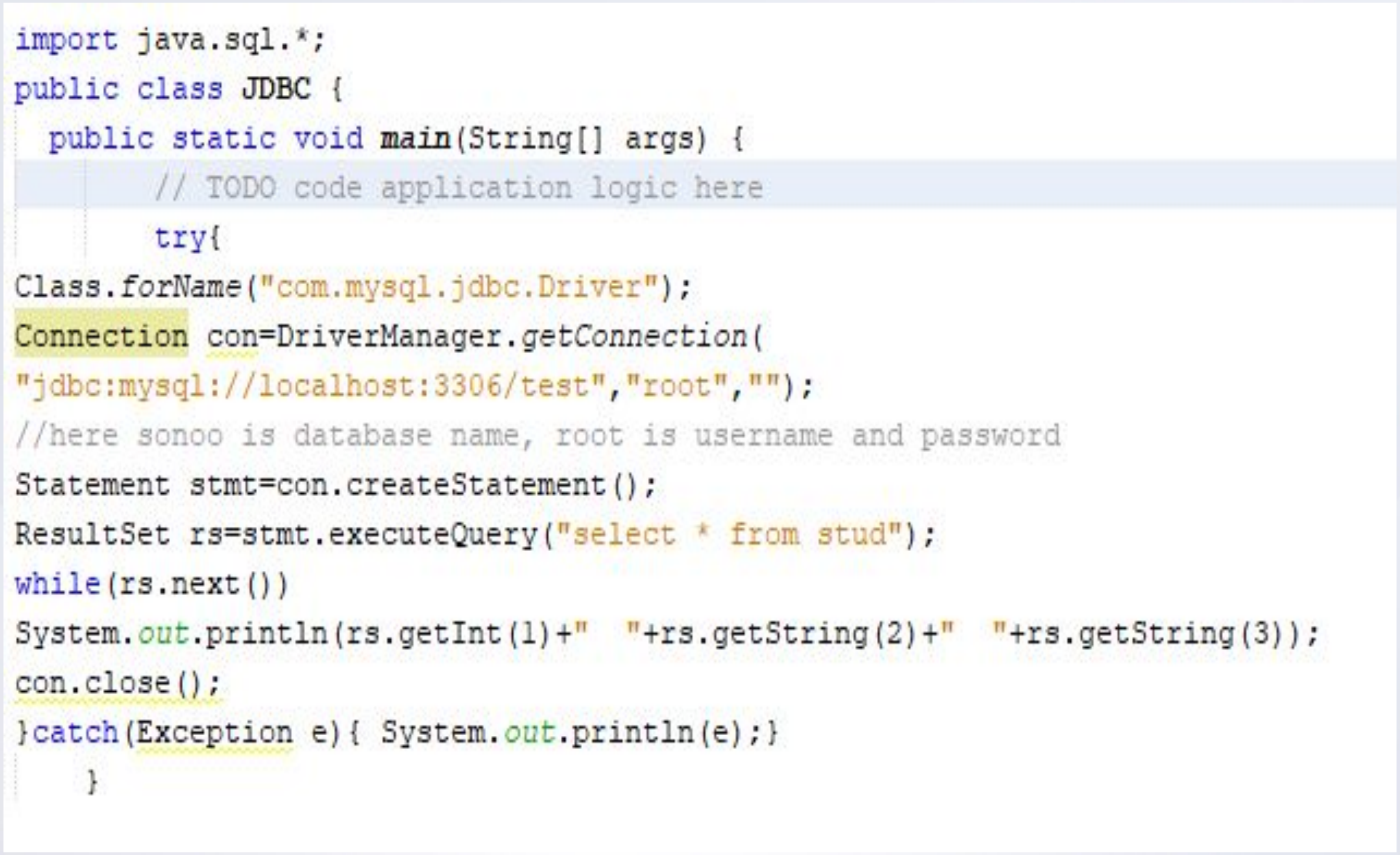
- It has overloaded signatures (parameters) which are as following:
  1. **public static Connection getConnection(String url) throws SQLExceptiion**
  2. **public static Connection getConnection(String url, String username, String password) throws SQLExceptiion**

  For example: getConnection (“jdbc: odbc: emp”) **OR**

  getConnection (“jdbc: odbc:emp”, “scott”,” tiger”);

Its returns the object of **Connection interface**, it throws **SQLException.**

Example



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

Statement Interface

1. **executeQuery( )**

   public ResultSet executeQuery(String sql) throws SQLException

   Used with **select query.**

2. **executeUpdate( )**

   public int executeUpdate(String sql) throws SQLException

   Used with **insert, update, delete, alter table** etc.

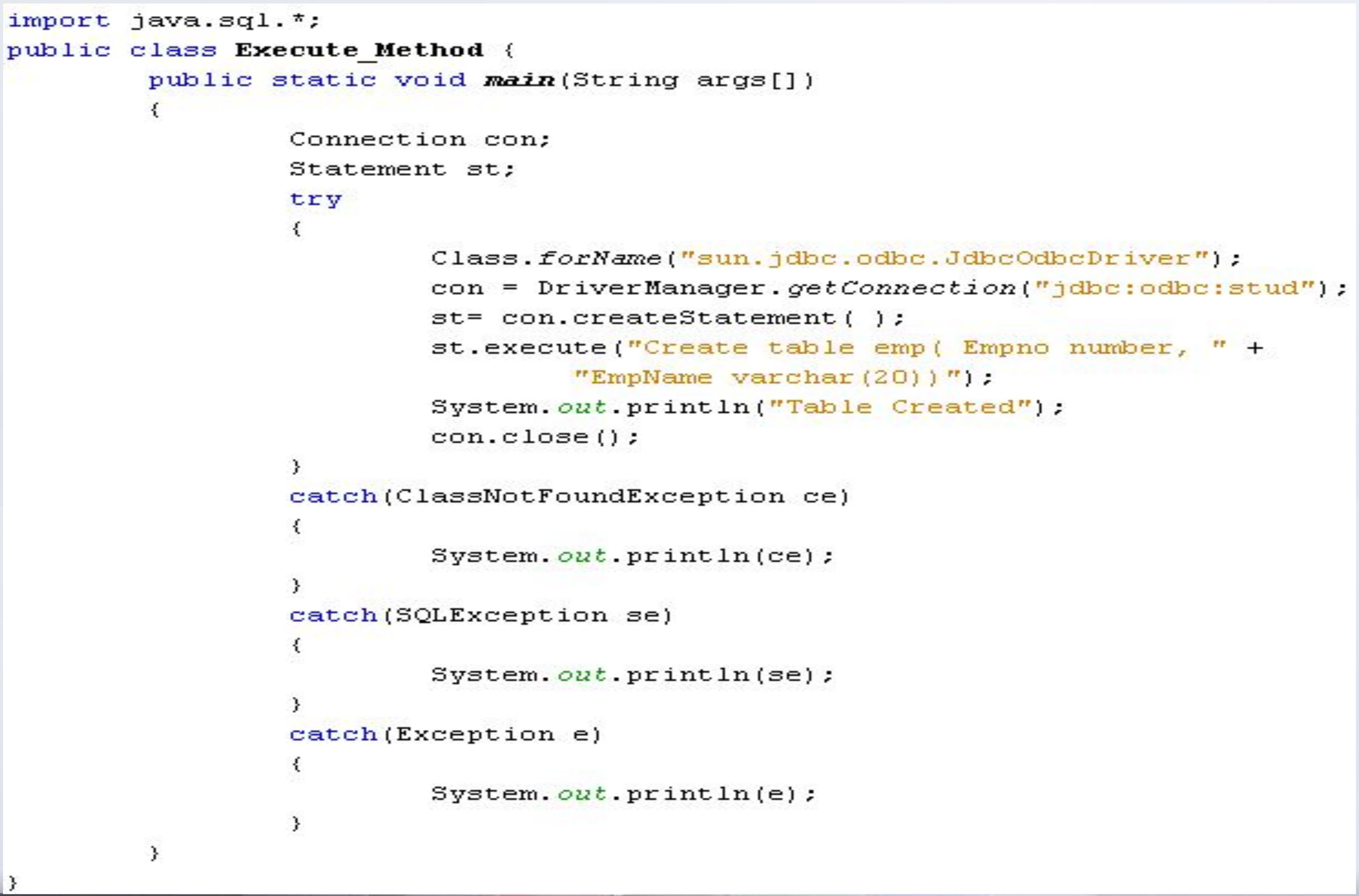
3. **execute( )**

   public boolean execute(String sql) throws SQLException

   Generally used with **multiple results are generated**.

   Also used with **Create table** query.

Ex of execute() of Statement



```
import java.sql.*;
public class Execute_Method {
        public static void main(String args[])
        {
                Connection con;
                Statement st;
                try
                {
                        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
                        con = DriverManager.getConnection("jdbc:odbc:stud");
                        st= con.createStatement( );
                        st.execute("Create table emp( Empno number, " +
                                "EmpName varchar(20))");
                        System.out.println("Table Created");
                        con.close();
                }
                catch(ClassNotFoundException ce)
                {
                        System.out.println(ce);
                }
                catch(SQLException se)
                {
                        System.out.println(se);
                }
                catch(Exception e)
                {
                        System.out.println(e);
                }
        }
}
```

Ex. of executeUpdate()

```
import java.sql.*;
public class Update_Method {
public static void main(String args[])
        {
                Connection con;
                Statement st;
                int ans;
                try
                {
                    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
                    con = DriverManager.getConnection("jdbc:odbc:stud");
                    st= con.createStatement( );
                    ans=st.executeUpdate("Insert into emp values(1,"+"'xyz'"+")");
                    System.out.println(ans+" Row(s) Created");
                    con.close();
                }
                catch(ClassNotFoundException ce)
                {
                        System.out.println(ce);
                }
                catch(SQLException se)
                {
                        System.out.println(se);
                }
                catch(Exception e)
                {
                        System.out.println(e);
                }
        }
}
```

Ex of executeQuery()

```
import java.sql.*;
public class ExQuery_Method
{
   public static void main(String a[])
        {
           Connection con;
           Statement stmt;
           ResultSet rs;
           try
           {
               Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
               con = DriverManager.getConnection("jdbc:odbc:stud");
               stmt = con.createStatement();
               rs = stmt.executeQuery("Select * from Emp");
               while(rs.next())
               {
                   System.out.println("Employee No. : " + rs.getInt(1));
                   System.out.println("Employee Name :" + rs.getString(2));
               }
               rs.close();
               stmt.close();
               con.close();
           }
           catch(ClassNotFoundException ce){System.out.println(ce);}
           catch(SQLException se) { System.out.println(se); }
           catch(Exception e) { System.out.println(e); }
        }
}
```

Query To Database using PreparedStatement

Create a object using prepareStatement Method of Connection Interface

Syntax :

public PreparedStatement prepareStatement(String sql) throws SQLException

Example :

```
PreparedStatement pst;

pst = con.prepareStatement("Select * from emp where empno=?" );
```

Query To Database using PreparedStatement

Merge all values in SQL query where ? is given

To merge value of ? we have to use **setXXX** methods of PreparedStatement. Like **getXXX** methods of ResultSet interface

There are also various setXXX methods to merge values according to data type of field.

Syntax :

setXXX(parameterIndex, parameterValue) OR

setXXX(parametername, parametervalue)

Example :

```
PreparedStatement pst;

pst = con.prepareStatement(“Select * from emp where empno=?” );

pst.setInt(1, 5);
```

Query To Database using PreparedStatement

1. **executeQuery( ):**

   public ResultSet executeQuery( ) throws SQLException

2. **executeUpdate( ):**

   public int executeUpdate( ) throws SQLException

3. **execute( ):**

   public boolean execute( ) throws SQLException

Example :

```
PreparedStatement pst;
ResultSet rs;
pst = con.prepareStatement("Select * from emp where empno=?" );
pst.setInt(1, 5);
rs= pst.executeQuery( );
```

Ex: insert query using PreparedStatement

```
import java.sql.*;
class PSTMTDemo
{
        public static void main(String args[])
        {
            Connection conn;
            PreparedStatement pstmt;
            try
            {
                    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
                    String url = "jdbc:odbc:emp";
                    conn = DriverManager.getConnection(url);
                    String ins = "Insert into emp values(?,?)";
                    pstmt = conn.prepareStatement(ins);
                    pstmt.setInt(1,1);
                    pstmt.setString(2,"vaishali");
                    pstmt.executeUpdate();
                    System.out.println("Record inserted");
                    conn.close();
            }
           catch(ClassNotFoundException ce){System.out.println(ce);}
           catch(SQLException se) { System.out.println(se); }
           catch(Exception e) { System.out.println(e); }
        }
}
```

Steps for CallableStatement

Step 1: Create a CallableStatement object.
Step 2: Pass values to the input (IN) parameters.
Step 3: Indicate which parameters are output-only (OUT) or input and output (INOUT) parameters.
Step 4: Call the stored procedure using any one the execute methods.
Step 5: If the stored procedure returns result sets, retrieve the result sets.
Step 6: Retrieve values from the OUT parameters or INOUT parameters.
Step 7: Close the CallableStatement object

Example of CallableStatement

Create a procedure

```
Create or replace procedure remove (name
varchar2) as
Begin
Delete from emp where
emp.Empname=name;
End;
```

Example of CallableStatement

Create a object using prepareCall Method of Connection Interface

Syntax :

public CallableStatement prepareCall(String sql) throws SQLException

Example :

```
CallableStatement cst;
cst = con.prepareCall("{call remove( ? )}" );
```

Example of CallableStatement

Merge all values in SQL query where ? is given

To merge value of ? we have to use **setXXX** methods of CallableStatement.
Syntax of setXXX methods:

```
setXXX(parameterIndex,parameterValue)
```

Example :

```
CallableStatement cst;
cst = con.prepareCall("{call remove( ? )}" );
cst.setString(1, "Dhruvi");
```

Query To Database using CallableStatement

Execute Query using method of CallableStatement

1. **executeQuery( ):**
   public ResultSet executeQuery( ) throws SQLException

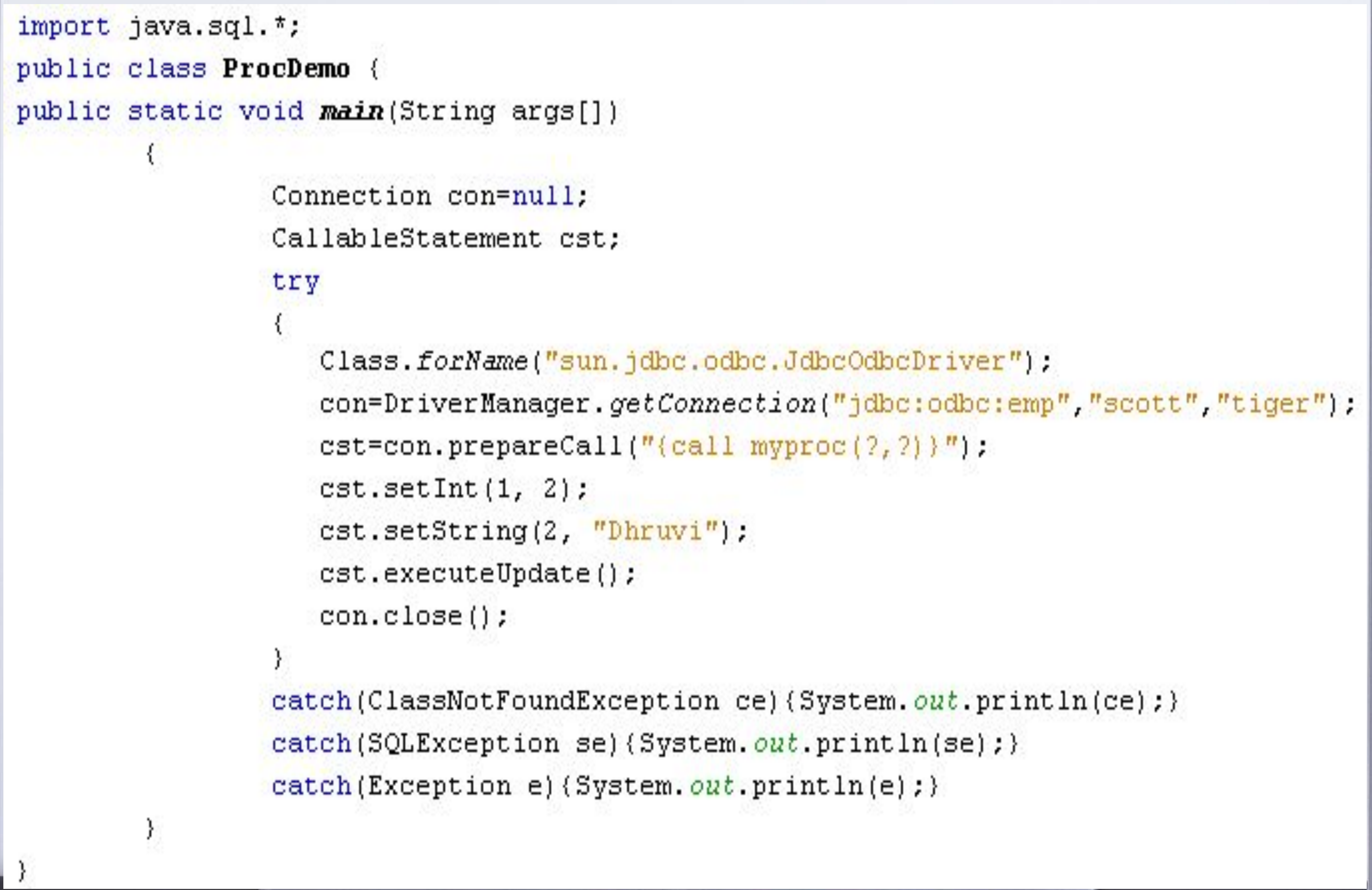
2. **executeUpdate( ):**
   public int executeUpdate( ) throws SQLException

3. **execute( ):**
   public boolean execute( ) throws SQLException
   Generally used with multiple results are generated.

Example :

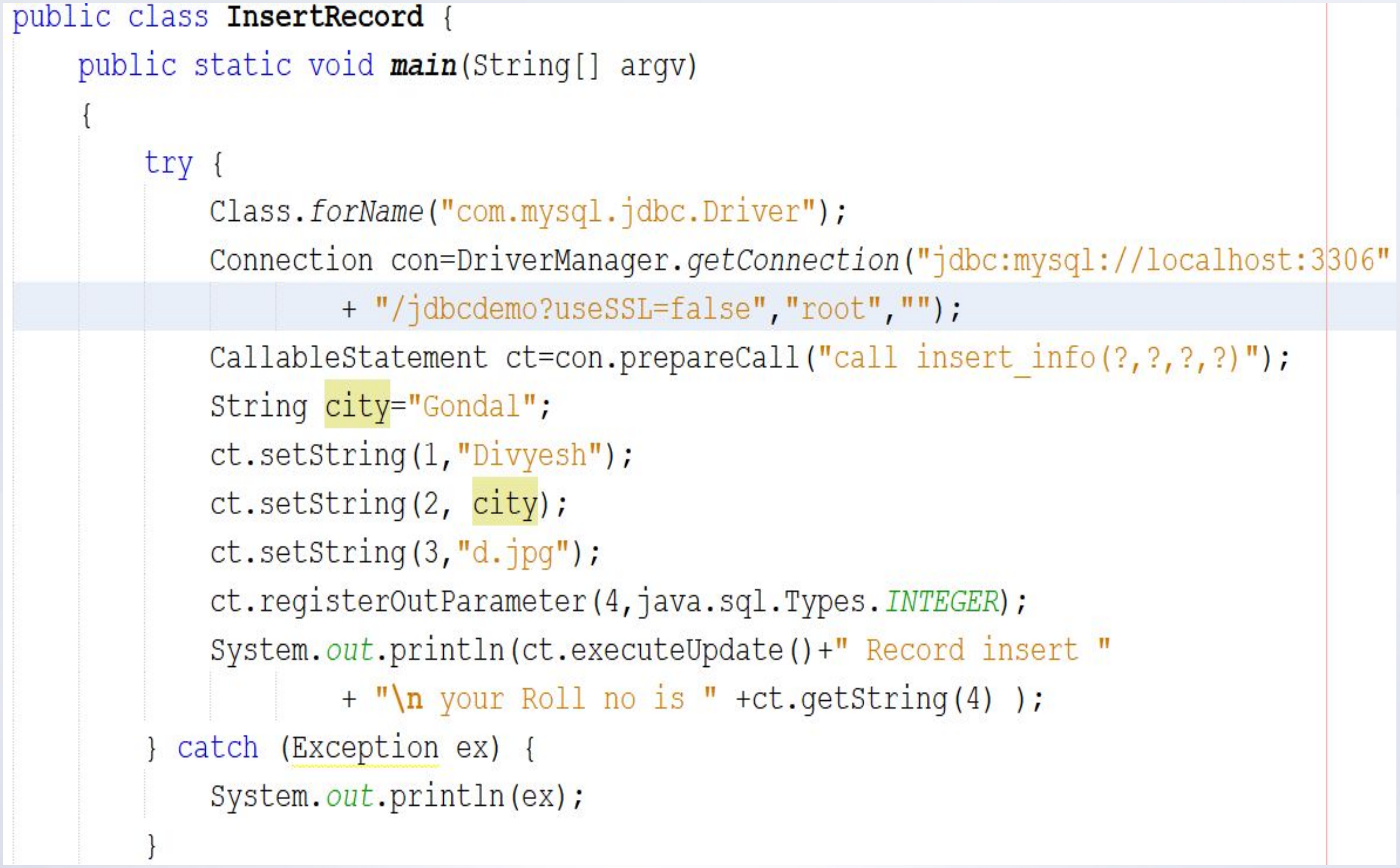
```
CallableStatement cst;
cst = con.prepareCall("{call remove( ? )}" );
cst.setString(1, "Dhruvi");
cst.executeUpdate( );
```

Example of CallableStatement



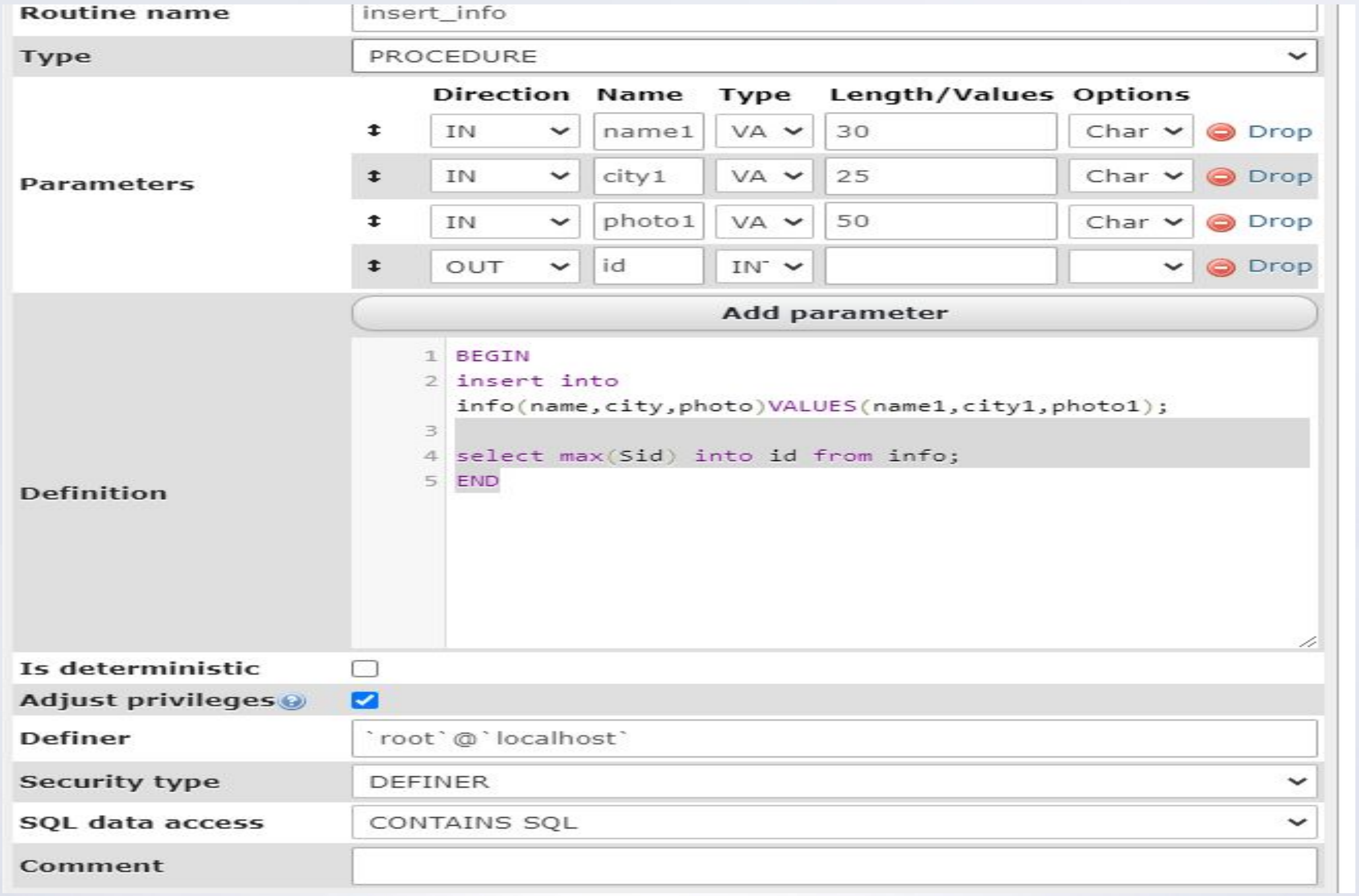
```
import java.sql.*;
public class ProcDemo {
public static void main(String args[])
        {
                Connection con=null;
                CallableStatement cst;
                try
                {
                   Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
                   con=DriverManager.getConnection("jdbc:odbc:emp","scott","tiger");
                   cst=con.prepareCall("{call myproc(?,?)}");
                   cst.setInt(1, 2);
                   cst.setString(2, "Dhruvi");
                   cst.executeUpdate();
                   con.close();
                }
                catch(ClassNotFoundException ce){System.out.println(ce);}
                catch(SQLException se){System.out.println(se);}
                catch(Exception e){System.out.println(e);}
        }
}
```

Example of IN and OUT Parameters



```
public class InsertRecord {
    public static void main(String[] argv)
    {
        try {
            Class.forName("com.mysql.jdbc.Driver");
            Connection con=DriverManager.getConnection("jdbc:mysql://localhost:3306"
                    + "/jdbcdemo?useSSL=false","root","");
            CallableStatement ct=con.prepareCall("call insert_info(?,?,?,?)");
            String city="Gondal";
            ct.setString(1,"Divyesh");
            ct.setString(2, city);
            ct.setString(3,"d.jpg");
            ct.registerOutParameter(4,java.sql.Types.INTEGER);
            System.out.println(ct.executeUpdate()+" Record insert "
                    + "\n your Roll no is " +ct.getString(4) );
        } catch (Exception ex) {
            System.out.println(ex);
        }
```

Example Procedure



Routine name	insert_info
Type	PROCEDURE

Parameters

	Direction	Name	Type	Length/Values	Options	
↕	IN	name1	VA	30	Char	Drop
↕	IN	city1	VA	25	Char	Drop
↕	IN	photo1	VA	50	Char	Drop
↕	OUT	id	IN			Drop

Add parameter

Definition

```
BEGIN
insert into
info(name,city,photo)VALUES(name1,city1,photo1);

select max(Sid) into id from info;
END
```

Is deterministic	☐
Adjust privileges	✓
Definer	\`root\`@\`localhost\`
Security type	DEFINER
SQL data access	CONTAINS SQL
Comment	

ResultSet Interface

Types of Result Sets

1. TYPE_FORWARD_ONLY:

   It defines that cursor from the current row can **move forward only**.
2. TYPE_SCROLL_INSETIVE:

   It defines that **cursor can scroll but can not be modified**.
3. TYPE_SCROLL_SENSETIVE:

   It defines that **cursor can scroll and also can be modified.**

Types of Concurrency:

1. CONCUR_READ_ONLY:

   It defines that ResultSet object can not be modified or updated.
2. CONCUR_UPDATABLE

   It indicates a result set that *can* be updated programmatically

Methods of ResultSet Interface

getXXX methods:

getString () ,getInt (), getBoolean (), getDouble(), getFloat(), getDate(), getLong(), getShort(), getByte(), getBlob()

Navigation methods:

first( ),previous( ), next( ), last ( ),
afterLast( ), beforeFirst( ),
relative(int row), absolute (int row).

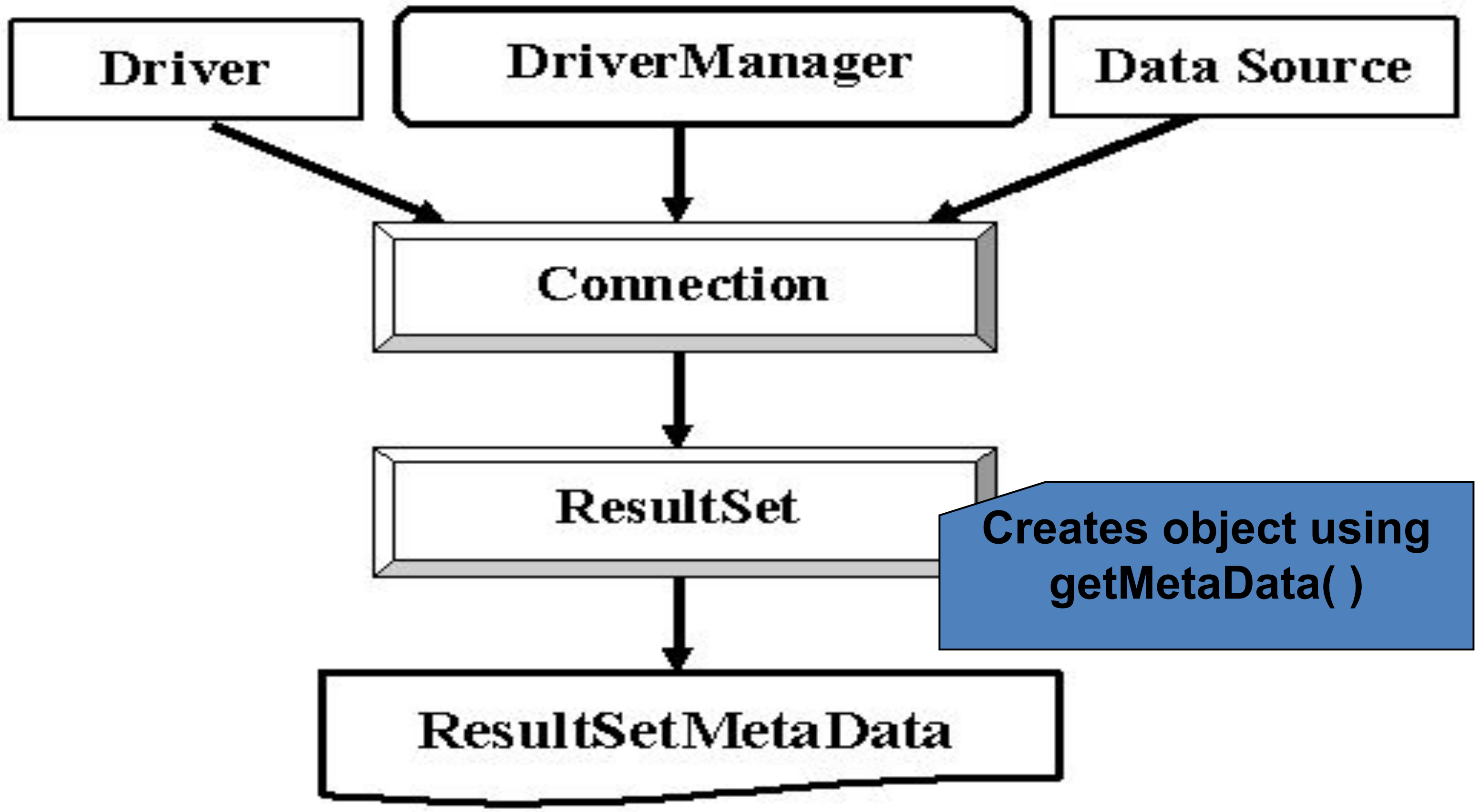
Other APIs

What is Metadata?

- **Metadata** is data about data (or information about information), which provides structured, descriptive information about other data.
- There are two types of MetaData interfaces are in java.sql package
  1. **java.sql.ResultSetMetaData Interface**
  2. **java.sql.DatabaseMetaData Interface**

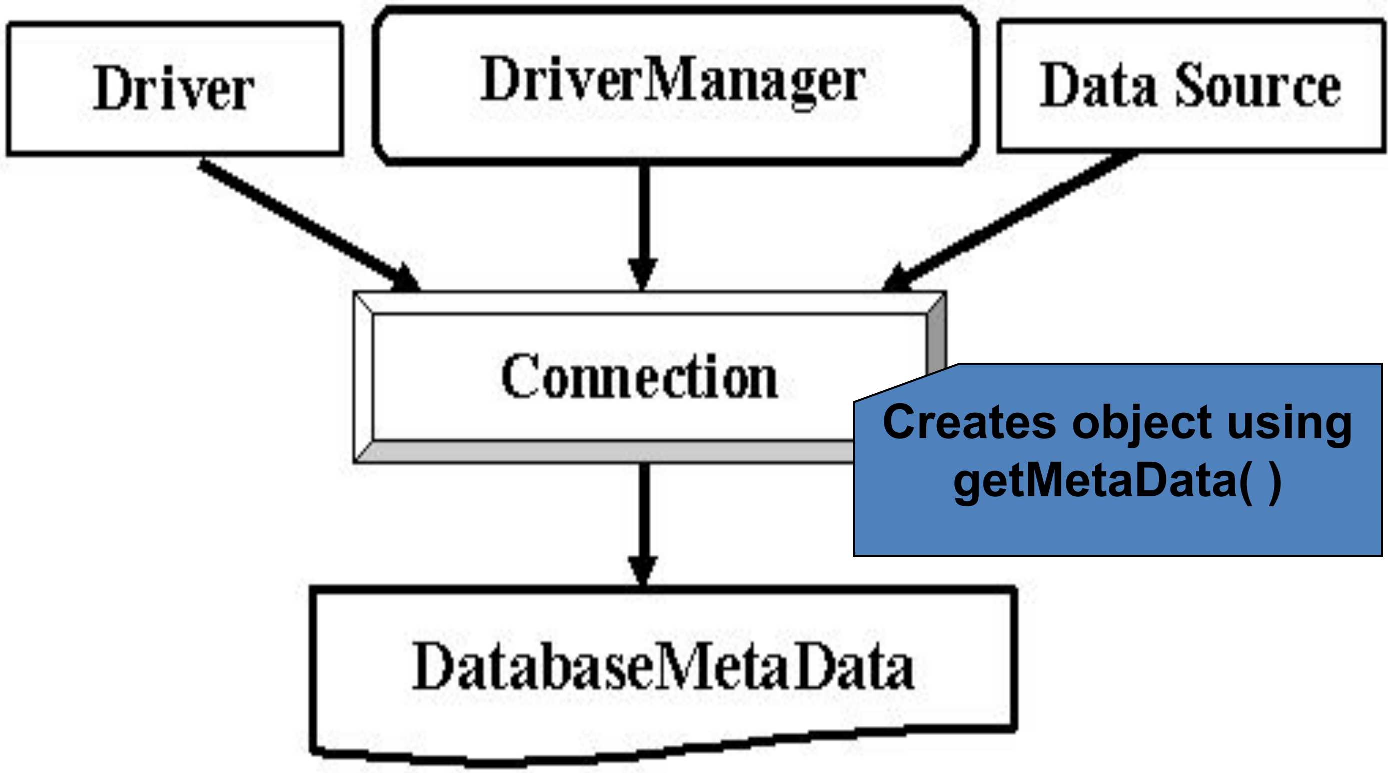
ResultSetMetaData Interface

Create a object of ResultSetMetaData Interface

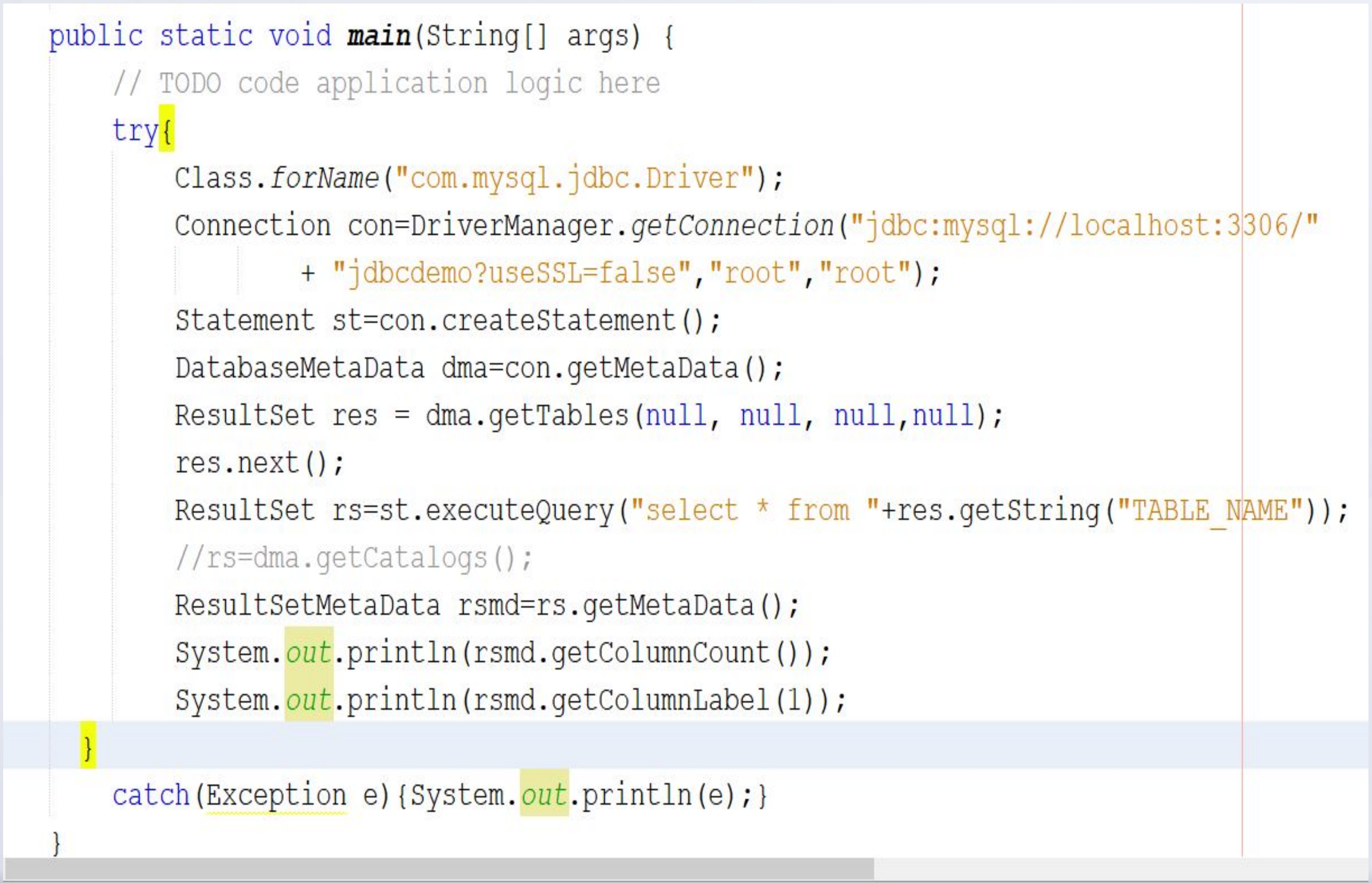


DatabaseMetaData Interface

Create a object of DatabaseMetaData Interface



Example of MetaData



```
public static void main(String[] args) {
    // TODO code application logic here
    try{
        Class.forName("com.mysql.jdbc.Driver");
        Connection con=DriverManager.getConnection("jdbc:mysql://localhost:3306/"
                + "jdbcdemo?useSSL=false","root","root");
        Statement st=con.createStatement();
        DatabaseMetaData dma=con.getMetaData();
        ResultSet res = dma.getTables(null, null, null,null);
        res.next();
        ResultSet rs=st.executeQuery("select * from "+res.getString("TABLE_NAME"));
        //rs=dma.getCatalogs();
        ResultSetMetaData rsmd=rs.getMetaData();
        System.out.println(rsmd.getColumnCount());
        System.out.println(rsmd.getColumnLabel(1));
  }
    catch(Exception e){System.out.println(e);}
}
```