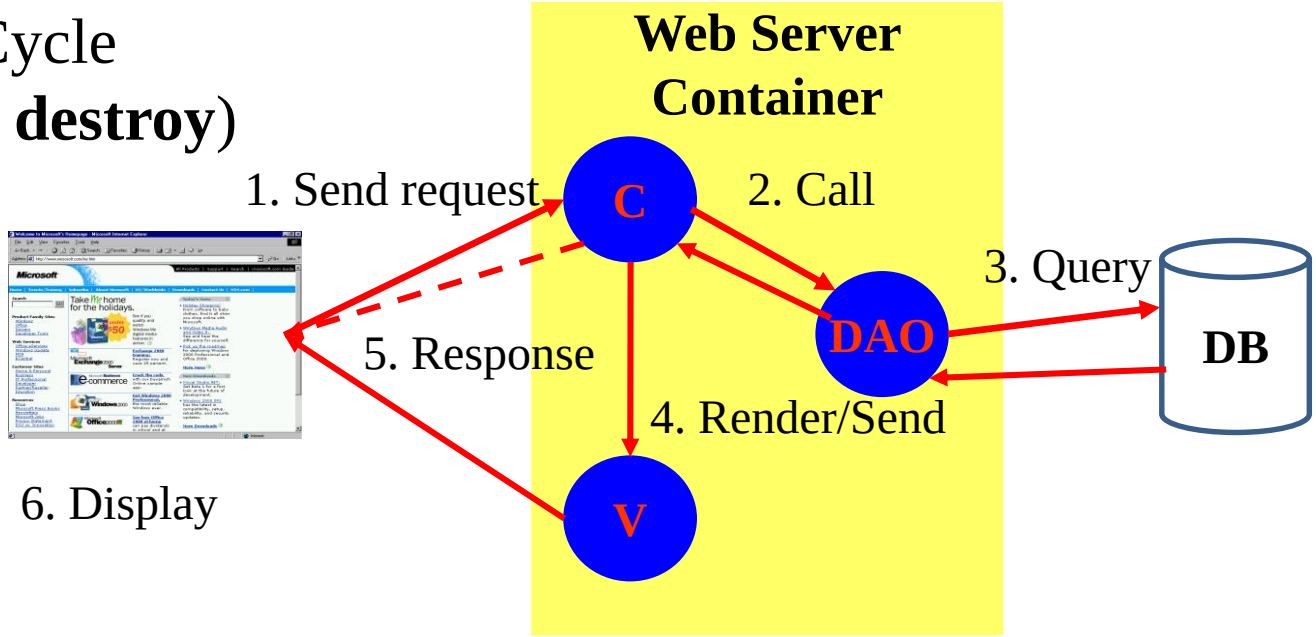
Fpt University

Web Application Interacting with Database

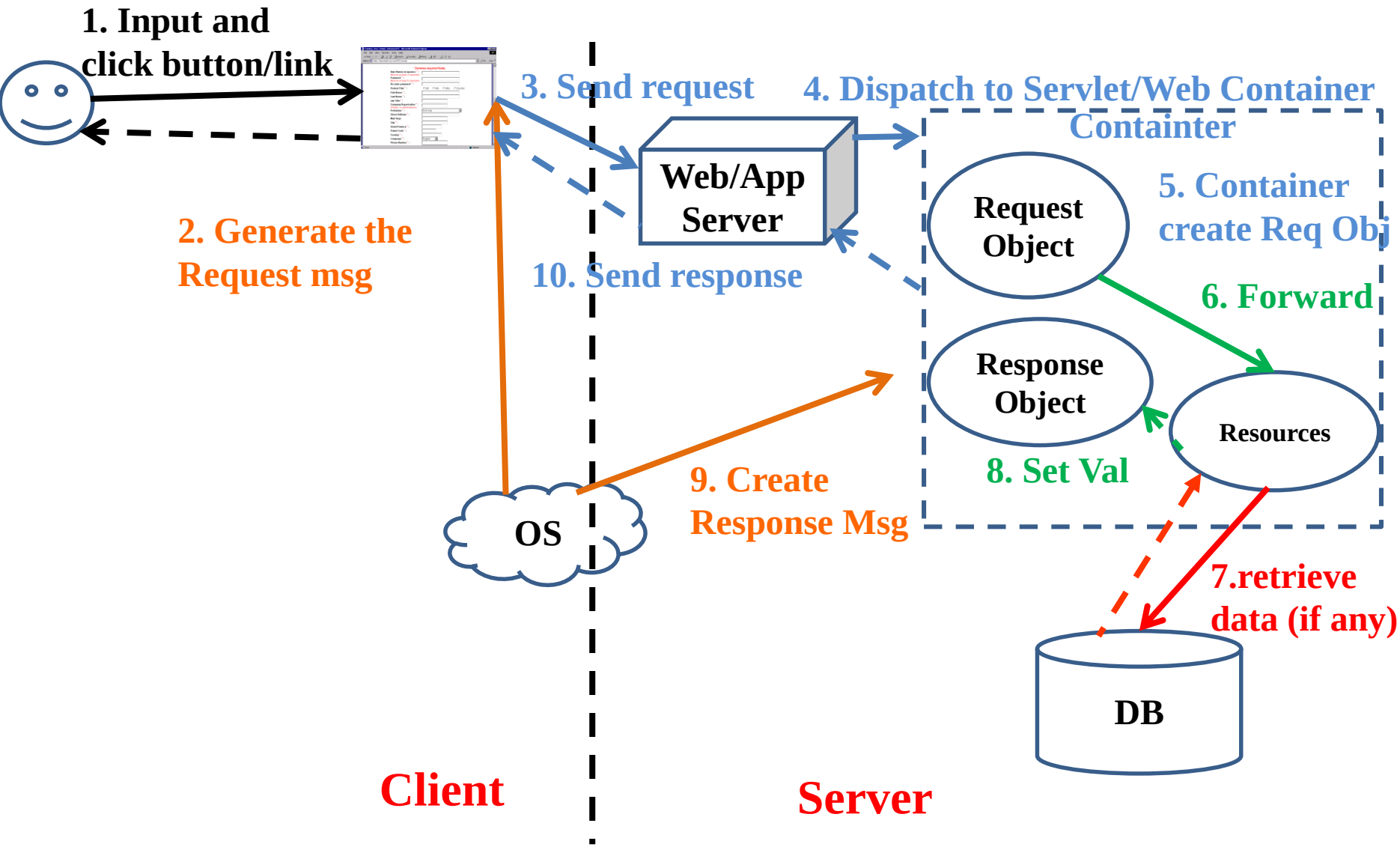
Java Database Connectivity

Review

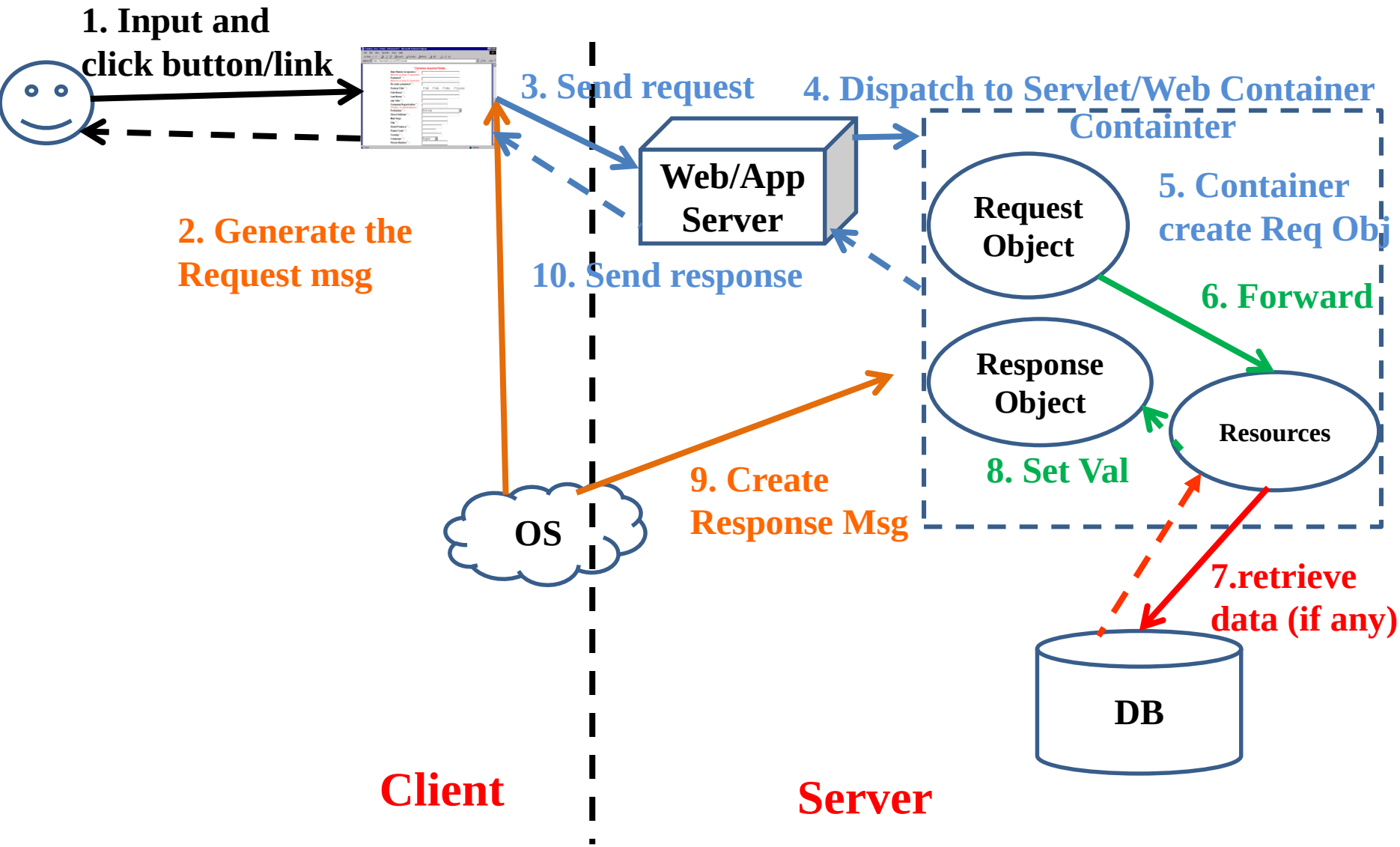
- **How to build the simple web site using html and servlet?**
  - Break down structure component in building web application
- **Some concepts**
  - Servlet vs. Java class, Parameter vs. Variable
  - Form Parameters
  - Http Protocol
  - HTTP Methods: **GET**, **POST**, …
  - Servlet Life Cycle (**init, service, destroy**)



Review


1. Input and click button/link
2. Generate the Request msg
3. Send request
4. Dispatch to Servlet/Web Container
Containter
Web/App Server
Request Object
5. Container create Req Obj
6. Forward
Response Object
Resources
8. Set Val
9. Create Response Msg
10. Send response
OS
7.retrieve data (if any)
DB
Client
Server

Review


1. Input and click button/link
2. Generate the Request msg
3. Send request
4. Dispatch to Servlet/Web Container
Containter
Web/App Server
Request Object
5. Container create Req Obj
6. Forward
Response Object
Resources
8. Set Val
9. Create Response Msg
10. Send response
OS
7.retrieve data (if any)
DB
Client
Server

Objectives

- **How to access database from web application?**
  - JDBC
  - Relational Database Overview
  - JDBC and JDBC Drivers
  - JDBC Basics: Processing SQL Statements
  - Implement CRUD application using MS SQL

Objectives

Web based App - PBL

Web

- Day 1, 2, 3 – Login Servlet ✓
  - 4, 5, 6 – JDBC Login
- Day 7, 8, 9 – Search Break Down
- Day 10, 11 – MVC2 JSP
- Day 12, 13, 14, 15 – CUD, Shopping Carts Sessions
- Day 16, 17 – Login JavaBeans
- Day 18, 19, 20 – CRUD MVC 2 Complete - JSTL - Taglib
- Day 21, 22, 23 – Filter MVC2 Using Filter as Controller

Overview
DB vs. DBMS

- **Databases**
  - Are **collection of related data** which are **stored** in **secondary** mass storage and are **used** by some processes **concurrently**.
  - Are **organized** in some ways in order to **reduce redundancies**.
- **DBMS: Database management system**
  - Is a **software** which manages some databases.
  - **Supports** ways to users/processes for creating, updating, manipulating on databases and security mechanisms are supported also.
  - **DBMS libraries** (C/C++ codes are usually used) support APIs for user programs to manipulate databases.

Overview

Relational DB

- Presents information in tables with **rows and columns**.
  - A table is referred to as a relation in the sense that it is a **collection of objects of the same type** (rows).
- A Relational Database Management System (**RDBMS**)
  - Handles the way data **is stored, maintained, and retrieved**.
  - **Ex:** MS Access, MS SQL Server, Oracle

KHANHKT\SQL2017.S...dbo.Registration

Column Name	Data Type	Allow Nulls
username	varchar(20)	☐
password	varchar(30)	☐
lastname	nvarchar(100)	☐
isAdmin	bit	☐

KHANHKT\SQL2017.S...dbo.Registration

username	password	lastname	isAdmin
hutruc	123456	Hu Truc	False
IA1301	123456	Class IA1301	False
khanh	kieu123	Khanh Kieu	True
khanh@Spring	123456	Spring annotati...	False
khanhSpring	123456	khanh	False
khanhSpringa	123456	dddssaaaa	False

RDBMS

Structure Query Language (SQL)

- Common DML – **Data Manipulating Language** queries.
  - **SELECT** columns **FROM** tables [**WHERE** condition]
  - **UPDATE** table **SET** column=value,… **WHERE** condition
  - **DELETE FROM** table **WHERE** condition
  - **INSERT INTO** table (col1, col2,…) **VALUES** (val1, val2,…)

JDBC
Overview

- Java Database Connectivity
  - Is an application programming interface (**API**) for the programming language **Java**, which defines how a **client** may **access a database**
  - Provides access to DB and performs DB operations – **CRUD**(**C**reate, **R**ead, **U**pdate, **D**elete)

JDBC
API

- The JDBC™ API
  - Was designed to keep simple things simple such as **executing** common **SQL** statements, and **performing** other **objectives** common to database applications
  - Is a Java API that can **access any kind** of tabular data, especially data stored in a **Relational Database.**
  - Executes **simple SQL queries** in the Java code to **retrieve data** from database
  - The **java.sql.*** and **javax.sql.*** package provides database access in Java through directly or indirectly (Data source – flexible), and provides classes and interfaces that are used to interact with the database

JDBC

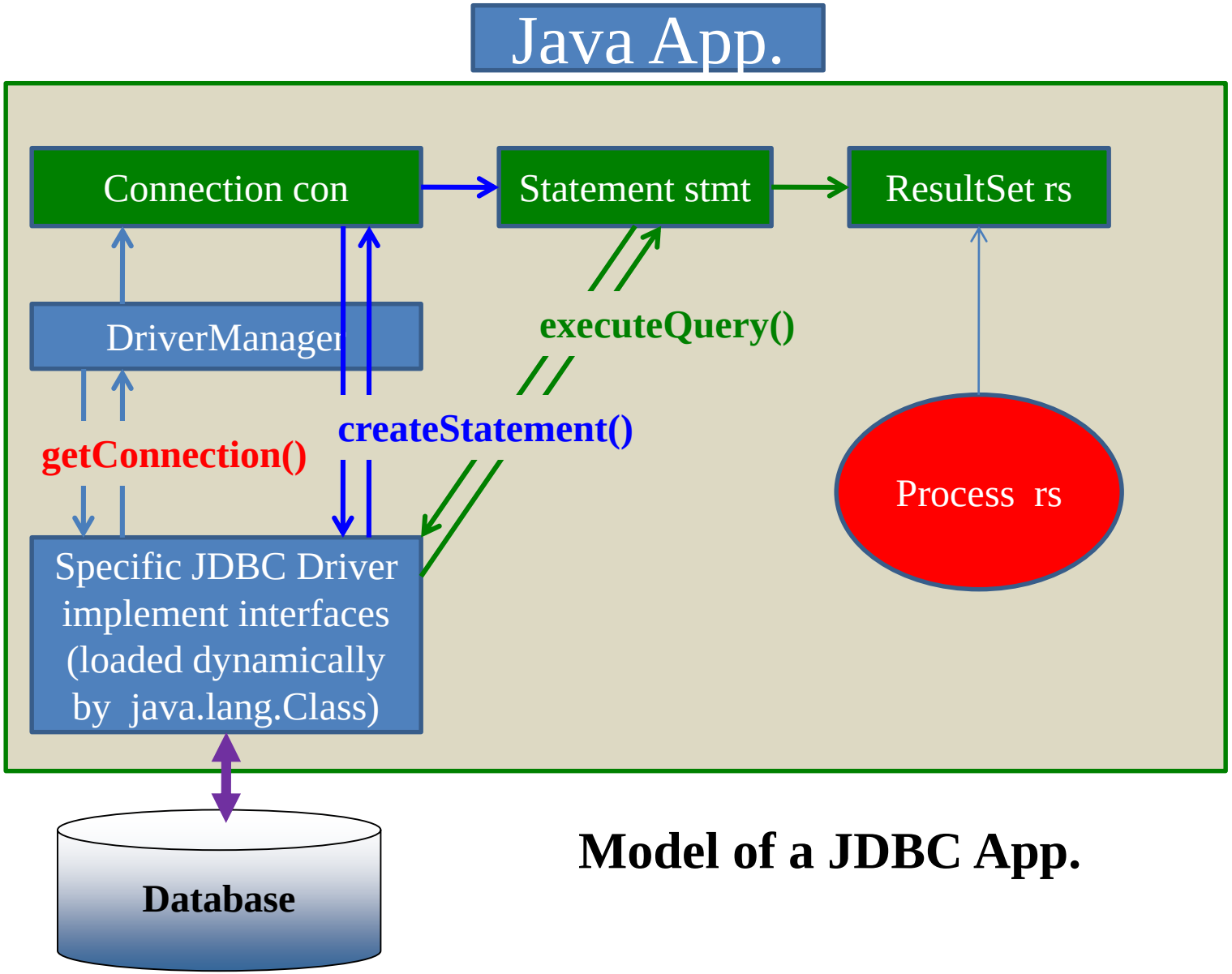
API

- JDBC APIs has 02 parts in the **java.sql** package

Part	Details	Purposes
JDBC Driver	**DriverManager** class	java.lang.Class.forName(DriverClass) will dynamically load the concrete driver class, provided by a **specific provider for a specific database**. This class implemented methods declared in JDBC interfaces. The class DriverManager will get a connection to database based on the specific driver class loaded.
JDBC API	Interfaces: **Connection,** **Statement** **ResultSet** **DatabaseMetadata** **ResultSetMetadata** Classes **SQLException**	 For creating a connection to a DBMS For executing SQL statements For storing result data set and achieving columns For getting database metadata For getting resultset metadata

Refer to the java.sql package for more details in Java documentation

JDBC API



Model of a JDBC App.

JDBC

API

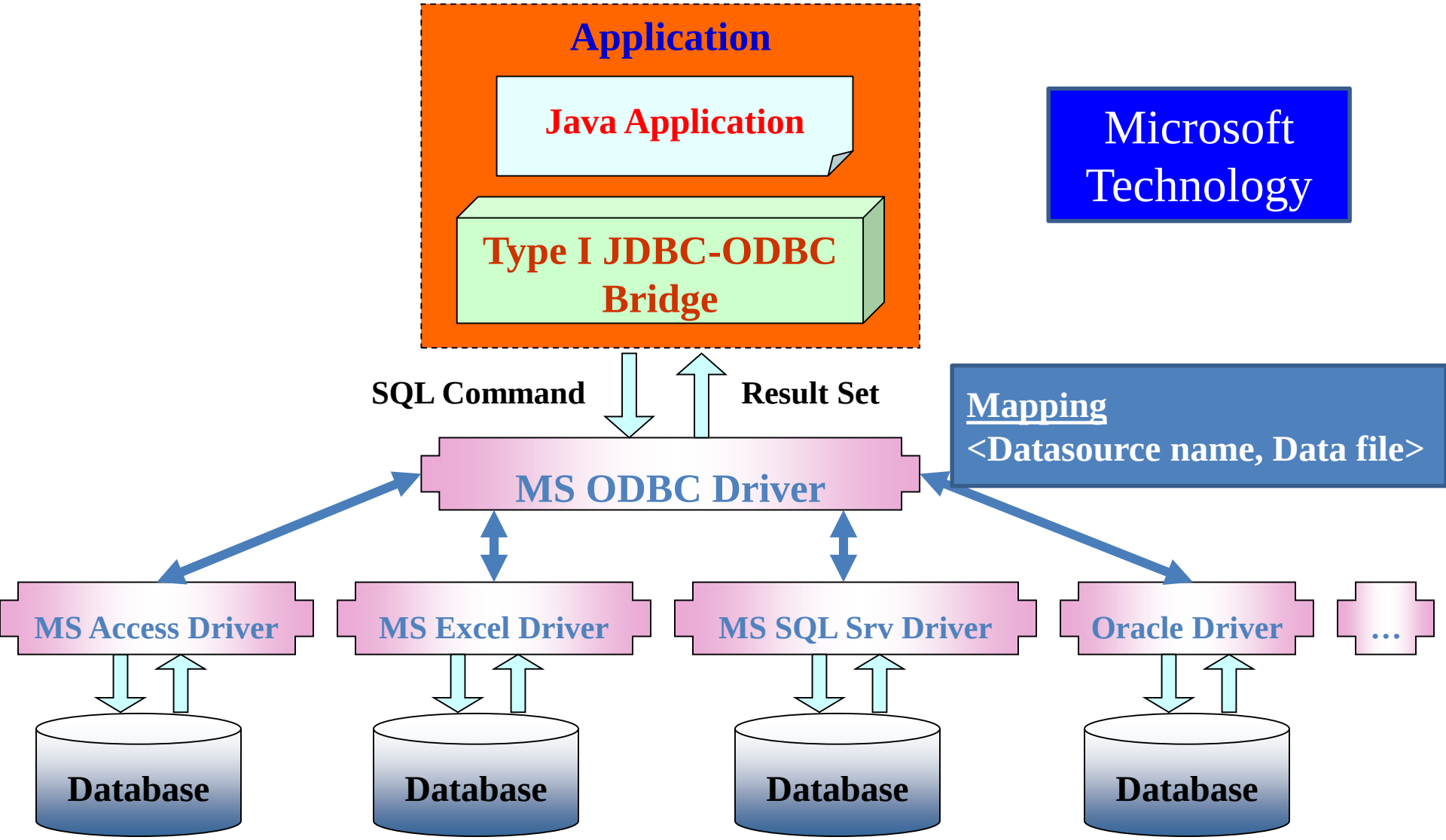
- **DBMS provider/developer** will supply a package in which **specific classes** implementing **standard JDBC driver** (**free**).
- Based on characteristics of DBMSs, **four types** of JDBC drivers are:
  - **Type 1**: JDBC ODBC
  - **Type 2**: Native API
  - **Type 3**: Network Protocol
  - **Type 4**: Native Protocol
- **Type 1 and Type 4** are **populated**.

JDBC
Drivers

- **Translates** Java statements to SQL statements
- Helps applications to **interact** with the database, using Java's built-in **Driver Manager**
- JDBC driver manager **maintains a list of drivers** created for different databases
- JDBC drivers connect the Java application to the driver **specified in the Java program**
- The **four types** of JDBC drivers are:
  - **Type 1**: JDBC ODBC
  - **Type 2**: Native API
  - **Type 3**: Network Protocol
  - **Type 4**: Native Protocol
  - **Type 1 & Type 4** is populated

JDBC

Type 1-Driver: JDBC-ODBC Bridge



JDBC

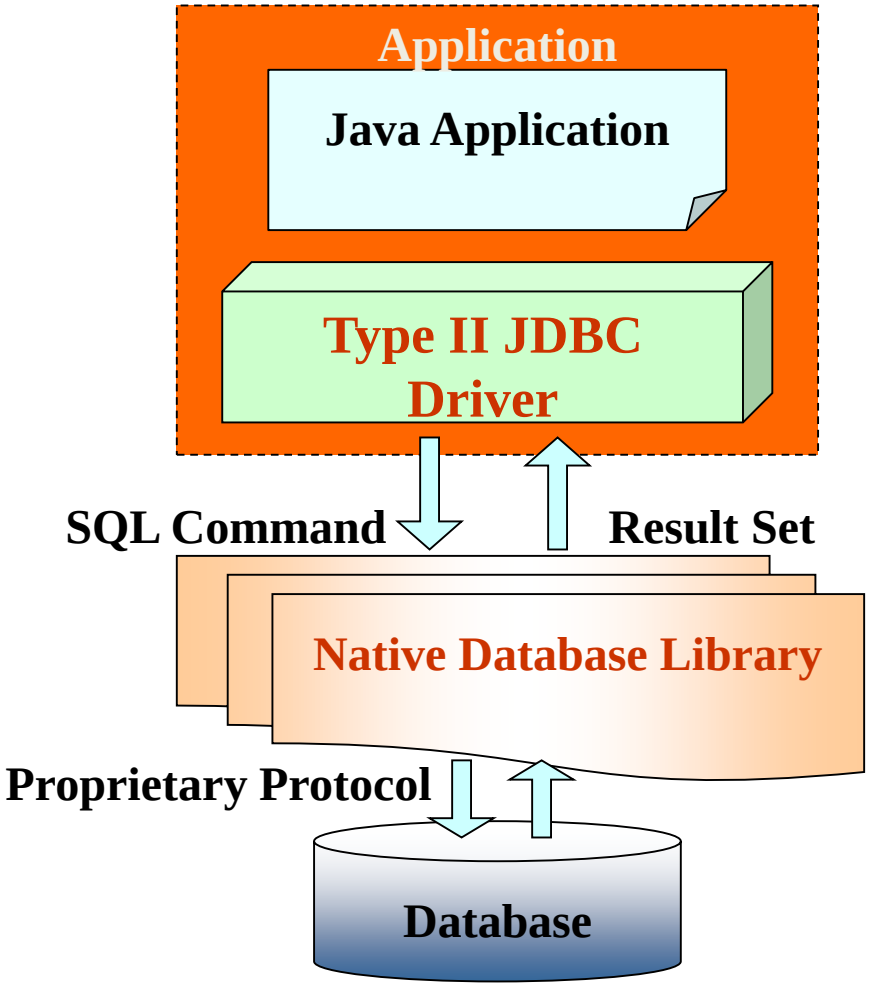
Type 1-Driver: JDBC-ODBC Bridge

- This package is in the JDK as default.
- **Translates** JDBC APIs **to** ODBC APIs
- **Enables** the Java applications to **interact** with any database supported by Microsoft.
- **Provides platform dependence**, as JDBC ODBC bridge driver uses ODBC
- **JDBC-ODBC bridge is useful when Java driver is not available for a database but it is supported by Microsoft.**
- **Disadvantages**
  - Platform **dependence** (Microsoft)
  - The performance is comparatively slower than other drivers
  - Require the ODBC driver and the client DB to be on the server.
- **Usage**: DSN is registered to use connecting DB (a data source is declared in Control Panel/ODBC Data sources)

JDBC

Type 2-Driver: Native API

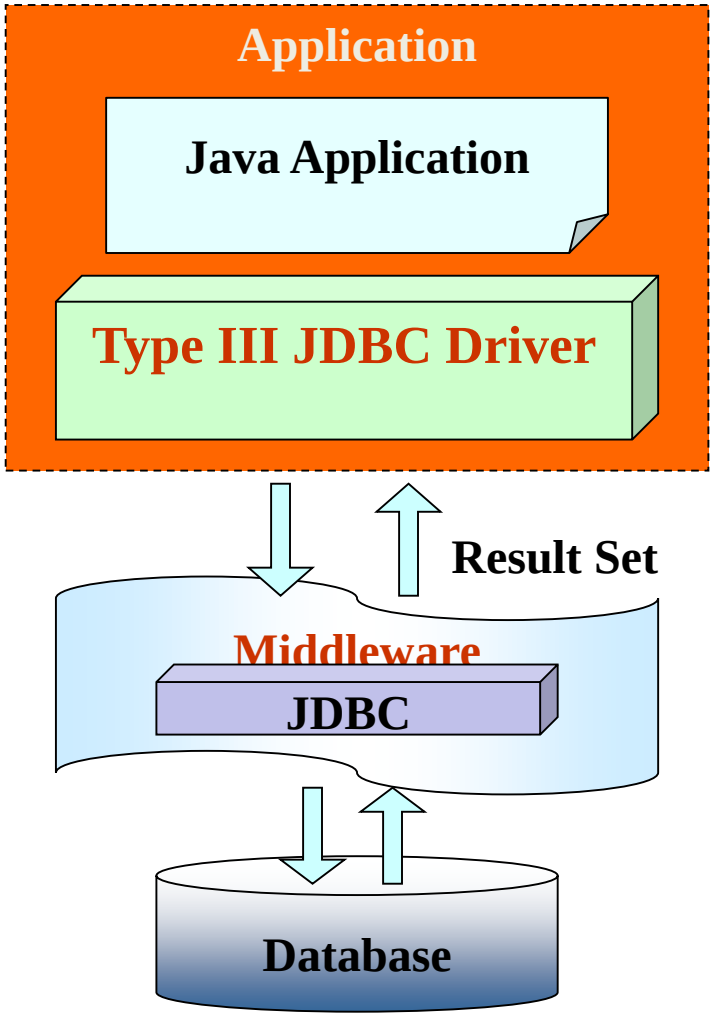
- Provides **access** to the database **through C/C++ codes.**
- Developed **using native code libraries**
- Native code libraries provide access to the database, and **improve the performance**
- Java application sends a request for database connectivity as a normal JDBC **call to the Native API driver**
- **Establishes** the call, and **translates** the call to the particular database protocol that is **forwarded** to the database



JDBC

Type 3-Driver: Network Protocol

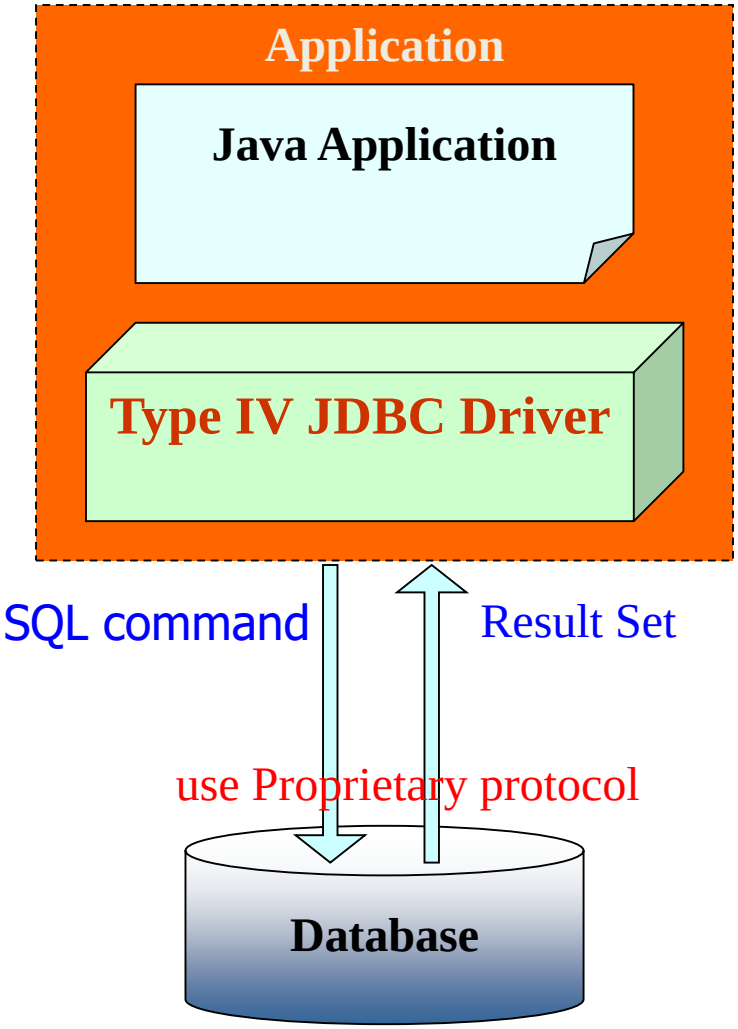
- Use a **pure Java client** and **communicate** with a **middleware server using** a **database-independent protocol**.
- The **middleware server** then **communicates** the **client's requests** to the data source
- **Manages multiple Java applications** connecting to different databases



JDBC

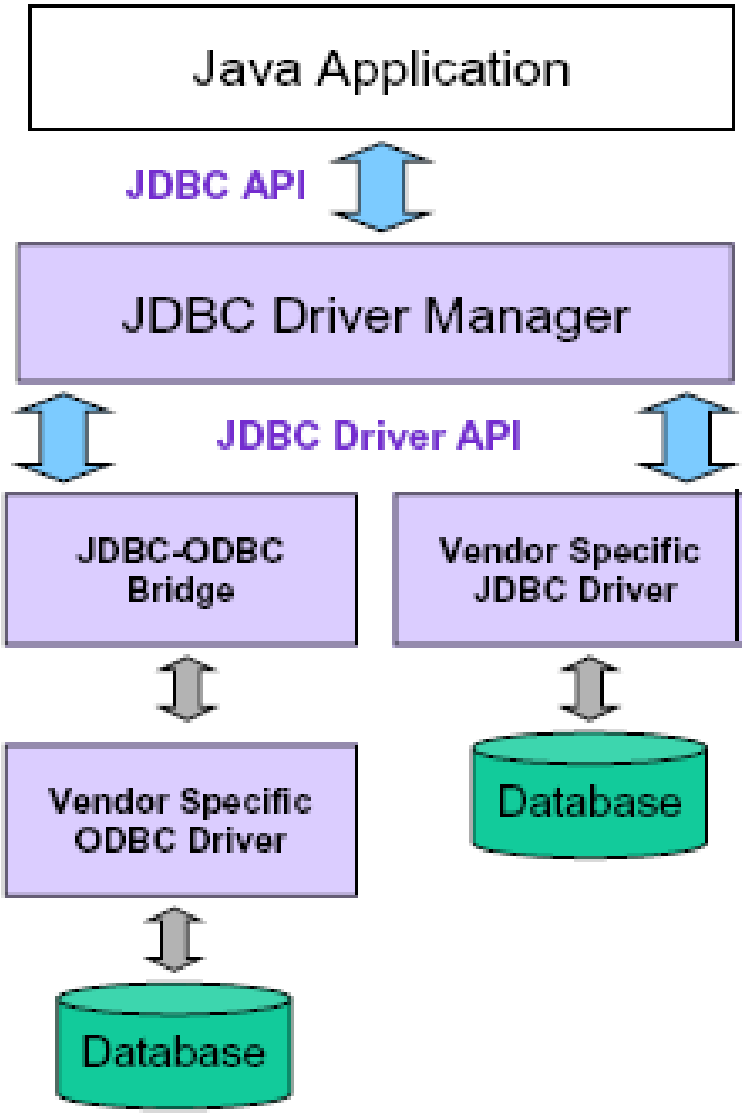
Type 4-Driver: Native Protocol

- **Communicates directly** with the database **using Java sockets**
- **Improves** the **performance** as translation is not required
- Converts JDBC queries into **native calls** used by the particular RDBMS
- The **driver library** is required when it is **used and attached** with the **deployed application** (**sqlserver 2000**: mssqlserver.jar, msutil.jar, msbase.jar; **sqlserver 2005**: sqljdbc.jar; **jtds**: jtds.jar …)
- **Independent platform**



JDBC

Summary



JDBC API

Download Driver & Configure RDBMS

- **Download:**
  - Download Microsoft JDBC Driver for SQL Server - SQL Server | Microsoft Docs
- **Configuration**
  - Using SQLServer Configuration Manager
  - Or, services.msc
  - Configure ports, protocol

JDBC API

- **How to connect DB**
  - Required
    - RDBMS: SQL Server
    - Driver Connection: sqljdbc4.jar
  - Steps
    - Load **Driver**
      - using **Class.forName method**
      - Driver string: **com.microsoft.sqlserver.jdbc.SQLServerDriver**
      - Exception: **ClassNotFoundException**
    - Create **connection String**
      - **protocol:server://ip:port;databaseName=DB[;instanceName=Instance]**
    - Open connection
      - **Connection con = DriverManager.getConnection(url, “user", “pass");**
      - Exception: **SQLException**

JDBC API

- **How to connect DB**
  - Implementation

DBUtils.java

```java
 * @author Trong Khanh
 */
public class DBUtils implements Serializable {

    public static Connection makeConnection() {
        try {
            //1. load driver
            Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
            //2. Tao connection string - protocol:loaiserver://diachiIP:port;tendb;teninstance
            String url = "jdbc:sqlserver://localhost:1433;databaseName=Sinhvien2K8;instanceName=SQL2008";
            //3. Lay Connection
            Connection con = DriverManager.getConnection(url, "sa", "trongkhanh");
            //4. Tra connection cho doi tuong goi
            return con;
        } catch (ClassNotFoundException ex) {
            ex.printStackTrace();
        } catch (SQLException ex) {
            ex.printStackTrace();
        }
        return null;

    }
}
```

JDBC API

- **How to access DB using JDBC API**
  - Required
    - DB is connected
  - Steps
    - Connect DB using method that you are built
      - **Check available DB connection**
    - Create **SQL String using DML**
    - Create Statement
      - Statement
      - **PrepareStatement (pass Parameter with ?)**
      - CallableStatement
    - **Execute** Query to **get ResultSet**
    - **Process** the ResultSet
  - **Notes**: **must closed** all objects that are created **after process had finished**

JDBC API

Statements

- **Sends** queries and command to the database
  - **Ex**: “Select * From Registration”
- **Is Created** from the Connection object
  - Statement stmt = con.createStatement();
  - Statement stmt = con.createStatement(rsType, rsConcurrency)
    - **rsType:** TYPE_FORWARD_ONLY, TYPE_SCROLL_INSENSITIVE, TYPE_SCROLL_SENSITIVE
    - **rsConcurrency**: CONCUR_READ_ONLY, CONCUR_UPDATABLE
  - There are **03 types** of Statement
    - Statement
    - PreparedStatement (prepareStatement). **Ex**:

      Select * From Registration Where uName = ?
    - CallableStatement (prepareCall()). **Ex**: {stpInsert (?)}

JDBC API

Prepared Statements

- To **execute** a Statement object **many times**, it normally **reduces execution time** to use a PreparedStatement object instead.

```
PreparedStatement stm = con.prepareStatement(sql);
```

- Supplying Values for PreparedStatement Parameters:
  - To **supply values to be used in place** of the question mark **placeholders** (if there are any) **before** a PreparedStatement object is **executed**.
  - One of the **set*XXX* methods** is called in the PreparedStatement class.

```
stm.setXXX(Cardinal number, values);
```

  - **Ex**:
    - stm.SetInt(1, 5);
    - stm.SetString(2, “abc”);

JDBC API

Callable Statements

- To **execute** the **stored procedure**, the Callable Statement is used
  - CallableStatement cs
    = con.prepareCall("{call stored_p_name(?)}");

JDBC API

- **How to access DB**
  - Implementation

```java
Connection con = null;
PreparedStatement stm = null;
ResultSet rs = null;

try {
    //1. connect DB
    con = DBHelper.makeConnection();
    //2. Create SQL String
    if (con != null) {
        String sql = "Select username "
                + "From Registration "
                + "Where username = ? And password = ?";
        //3. Create Statement Object
        stm = con.prepareStatement(sql);
        stm.setString(1, username);
        stm.setString(2, password);
        //4. Execute Query
        rs = stm.executeQuery();
        //5. process result
        if (rs.next()) {
            return true;
        }//end if
    }//end if connection is opened
```

```java
} finally {
    if (rs != null) {
        rs.close();
    }
    if (stm != null) {
        stm.close();
    }
    if (con != null) {
        con.close();
    }
}
```

JDBC API

- **How to access DB using**
  - Implementation

```java
        try {
            //1. connect DB
            con = DBHelper.makeConnection();
            //2. Create SQL String
            if (con != null) {
                String sql = "Delete From Registration "
                        + "Where username = ?";
                //3. Create Statement Object
                stm = con.prepareStatement(sql);
                stm.setString(1, username);
                //4. Execute Query
                int row = stm.executeUpdate();
                //5. process result
                if (row > 0) {
                    return true;
                }
            }//end if connection is opened
        } finally {
```

JDBC API

- **How to access DB**
  - Implementation

```
158                 //1. connect DB
159                 con = DBHelper.makeConnection();
160                 //2. Create SQL String
161                 if (con != null) {
162                     String sql = "Insert Into "
163                             + "Registration(username, password, lastname, isAdmin) "
164                             + "Values(?, ?, ?, ?)";
165                     //3. Create Statement Object
166                     stm = con.prepareStatement(sql);
167                     stm.setString(1, username);
168                     stm.setString(2, password);
169                     stm.setString(3, fullName);
170                     stm.setBoolean(4, role);
171
172                     //4. Execute Query
173                     int row = stm.executeUpdate();
174                     //5. process result
175                     if (row > 0) {
176                         return true;
177                     }
178                 }//end if connection is opened
179             } finally {
```

JDBC API

Execute Query

- Used to **execute** statement and **get data** from DB
  - The **executeQuery()** method
    - Is used to Query commands and stored procedure. **Ex**:

    String strSQL = “Select * From Registration”;
    - Returns **an object of type ResultSet**

    ResultSet rs = stmt.executeQuery(strSQL);
  - The **executeUpdate()** method
    - Is used to Insert, Update, or Delete commands. **Ex**

    String strSQL = “Insert into Registration Values(“Aptech”, “Aptech”);
    - Returns the **row of executed validation**.

    int nRow = stmt.executeUpdate(strSQL);
  - The **execute**() method
    - Is use to create and delete DB objects as table, DB ...**Ex**:

    String strSQL = “Drop table Registration”;

    stmt.execute(strSQL);

JDBC API

Process the Results

- The **ResultSet class** implements a **collection of type Set** and allows to use it to **process one row at the time**
- Apply to the ResultSet object
  - The **getXxx** (cardinal number/ field name string) of the ResultSet object is used to get the field value.
    - Cardinal number starts with 1
    - Xxx is a DataType of the selected field
  - **The next()** method of the ResultSet object is used to process the results from the DB
  - **Ex**: while(rs.next()) *(Point the cursor next row)* {
    rs.getInt(1) or rs.getInt("userId");
    rs.getString(1) or rs.getString("username"); }
  - **Notes: The field must be accessed in the order**
  - The 2D Resultset support the access methods to DB as first, isFirst, last, ...
- There is a class **ResultSetMetaData** that helps to **determine** the **number, names and types of column** in the ResultSet
- **Close the connection after used**: The close() method is used. **Ex**: con.close();

JDBC API

Some methods of ResultSet

Methods	Description
getString()	Takes column number as an argument and returns value from the specified column number as a string to the ResultSet object
getInt()	Takes column number as an argument and returns the value from the specified column number as an integer to the ResultSet object
getFloat()	Takes column number as an argument and returns the value from the specified column number as float type to the ResultSet object
getDate()	Takes column number as an argument and returns the value from the specified column number as java.sql.Date to the ResultSet object
findColumn()	Takes a column name as a string parameter and returns the column index of the specified column name
wasNull()	Returns true if the last column value read was SQL NULL
getMetaData()	Returns the information about the columns of ResultSet object in a ResultSetMetaData class

JDBC API

```
76          //1. connect DB
77          con = DBHelper.makeConnection();
78          //2. Create SQL String
79          if (con != null) {
80              String sql = "Select username, password, lastname, isAdmin "
81                      + "From Registration "
82                      + "Where lastname Like ?";
83              //3. Create Statement Object
84              stm = con.prepareStatement(sql);
85              stm.setString(1, "%" + searchValue + "%");
86              //4. Execute Query
87              rs = stm.executeQuery();
88              //5. process result
89              while (rs.next()) {
90                  String username = rs.getString("username");
91                  String password = rs.getString("password");
92                  String fullname = rs.getString("lastname");
93                  boolean role = rs.getBoolean("isAdmin");
94
95                  RegistrationDTO dto = new RegistrationDTO(
96                          username, password, fullname, role);
97
98                  if(this.accountList == null) {
99                      this.accountList = new ArrayList<>();
100                 }
101
102                 this.accountList.add(dto);
103             } //end while
104         }//end if connection is opened
105     } finally {
```

JDBC API

Some method of ResultSetMetaData

Methods	Syntax	Description
getColumnCount()	int getColumnCount()	Returns the number of columns in the ResultSet object
getColumnName()	String getColumnName (int **column**)	Takes column number as a parameter and returns the designated column name
getColumnType()	int getColumnType (int **column**)	Takes column number as a parameter and returns the designated column's SQL type from java.sql.Types. The types include Array, char, Integer, Date, and Float
isReadOnly()	boolean isReadOnly (int **column**)	Takes the column number as a parameter and returns true if the designated column is not writable
isSearchable()	Boolean isSearchable (int **column**)	Takes column number as a parameter and returns true if the specified column can be used in where clause.
isNullable()	int isNullable (int **column**)	Returns the nullability status of the specified column. The nullability status includes, columnNullable, columnNoNulls, and columnNullableUnknown

Summary

- **How to access database from web application?**
  - JDBC
  - Relational Database Overview
  - JDBC and JDBC Drivers
  - JDBC Basics: Processing SQL Statements
  - Implement CRUD application using MS SQL

Q&A

Exercises

- Do it again all of demos
- Using servlet to write the programs as the following requirement
  - Present the Login form (naming LoginServlet) with title Login, header h1 – Login, 02 textbox with naming txtUser and txtPass, and the Login button
    - Rewrite above Login application combining with DB
  - Writing the ColorServlet that presents “Welcome to Servlet course” with yellow in background and red in foreground
  - Writing the ProductServlet includes a form with a combo box containing Servlet & JSP, Struts & JSF, EJB, XMJ, Java Web Services, and the button with value Add to Cart

Next Lecture

- **How to deploy the Web Application to Web Server?**
  - Web applications Structure
  - Request Parameters vs. Context Parameters vs. Config/Servlet Parameters
  - Application Segments vs. Scope
- **How to transfer from resources to others with/without data/objects?**
  - Attributes vs. Parameters vs. Variables
  - Redirect vs. RequestDispatcher
  - RequestDispatcher vs. Filter

Next Lecture

Web based App - PBL

Web

- Day 1, 2, 3, 4, 5, 6 – Login Servlet, JDBC ✓
- Day 7, 8, 9 – Search Break Down
- Day 10, 11 – MVC2 JSP
- Day 12, 13, 14, 15 – CUD, Shopping Carts Sessions
- Day 16, 17 – Login JavaBeans
- Day 18, 19, 20 – CRUD MVC 2 Complete - JSTL - Taglib
- Day 21, 22, 23 – Filter MVC2 Using Filter as Controller

Appendix – Build The Simple Web

Login Page

login.html

Source History

```html
<!DOCTYPE html>
...5 lines
<html>
    <head>
        <title>Login</title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
    </head>
    <body>
        <h1>Login Page</h1>

        <form action="SE1162Servlet" method="POST">
            Username <input type="text" name="txtUsername" value="" /><br/>
            Password <input type="password" name="txtPassword" value="" /><br/>
            <input type="submit" value="Login" name="btAction" />
            <input type="reset" value="Reset" />
        </form>
```

Appendix – Build The Simple Web

Invalid Page

invalid.html

Source History

```
1   <!DOCTYPE html>
2   ...5 lines
7   <html>
8       <head>
9           <title>Invalid</title>
10          <meta charset="UTF-8">
11          <meta name="viewport" content="width=device-width, initial-scale=1.0">
12      </head>
13      <body>
14          <h1>
            <font color="red">
16              Invalid username or password!!!
17          </font>
18          </h1>
19
20          <a href="login.html">Click here to try again</a><br/>
```

Appendix – Build The Simple Web

Search Page

search.html

Source History

```
<!DOCTYPE html>
...5 lines
<html>
    <head>
        <title>Search</title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
    </head>
    <body>
        <h1>Search Page</h1>
        <form action="SE1162Servlet">
            Search Value <input type="text" name="txtSearchValue" value="" /><br/>
            <input type="submit" value="Search" name="btAction" />
        </form>
    </body>
</html>
```

Appendix – Build The Simple Web Servlet

SE1162Servlet.java

Source History

```
23  * @author kieukhanh
24  */
25 public class SE1162Servlet extends HttpServlet {
26     private final String searchPage = "search.html";
27     private final String invalidPage = "invalid.html";
28     /** Processes requests for both HTTP <code>GET</code> and <code>POST</code> ...9 lines *
37     protected void processRequest(HttpServletRequest request, HttpServletResponse response)
38             throws ServletException, IOException {
39         response.setContentType("text/html;charset=UTF-8");
40         PrintWriter out = response.getWriter();
41         try {
42             String button = request.getParameter("btAction");
43             String url = invalidPage;
44             if (button.equals("Login")) {
45                 String username = request.getParameter("txtUsername");
46                 String password = request.getParameter("txtPassword");
47
48                 RegistrationDAO dao = new RegistrationDAO();
49                 boolean result = dao.checkLogin(username, password);
50
51                 if (result) {
52                     url = searchPage;
53                 }
54             }
55             response.sendRedirect(url);
56         } catch (NamingException ex) {
            ex.printStackTrace();
58         } catch (SQLException ex) {
            ex.printStackTrace();
60         } finally {
61             out.close();
```

Appendix – Build The Simple Web DAO

RegistrationDAO.java

Source History

```java
 * @author kieukhanh
 */
public class RegistrationDAO implements Serializable {
    public boolean checkLogin(String username, String password)
        throws SQLException, NamingException {
        Connection con = null;
        PreparedStatement stm = null;
        ResultSet rs = null;
        try {
            con = DBUtils.makeConnection();
            if (con != null) {
                String sql = "Select * From Registration Where username = ? And password = ?";

                stm = con.prepareStatement(sql);
                stm.setString(1, username);
                stm.setString(2, password);

                rs = stm.executeQuery();
                if (rs.next()) {
                    return true;
                }
            }
        } finally {
            if (rs != null) {
                rs.close();
            }
            if (stm != null) {
                stm.close();
            }
            if (con != null) {
                con.close();
            }
        }

        return false;
    }
```