

JDBC

Java Database Connectivity

Objectives

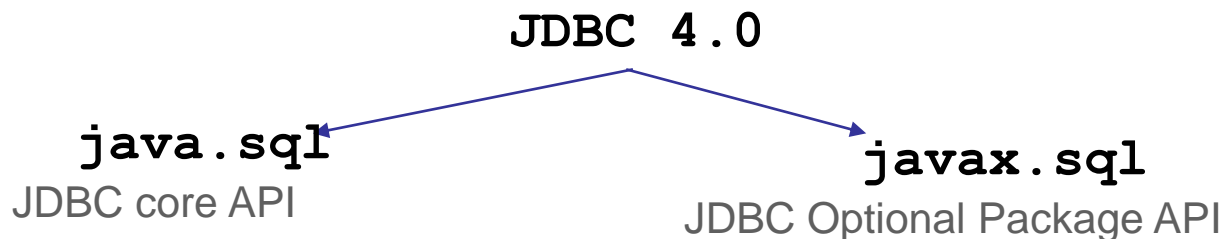
- Differentiate between database drivers and know when to use what,
- Write Java code to work with RDBMS databases to retrieve, insert, update and delete data

Table of Content

JDBC API	ResultSet methods (default)
java.sql.Connection	Advanced ResultSet
Driver Types	PreparedStatement
JDBC-ODBC Bridge	Inserting large objects in the database
Type2- Part Java, Part Native Driver	PreparedStatement for batch updates
Setting classpath to MySQL driver in eclipse	CallableStatement
JDBC 4.0 way to connect to database	Java Code to call the stored procedure
SQLException	Get ResultSet from stored procedure

JDBC API

- JDBC 4.0 (part of JSE 6) is an API, that provides standard for connectivity to a variety of data sources like SQL databases, spreadsheets and flat.
- JDBC provides call-level API for SQL databases; meaning, API defines a set of interfaces and abstract methods. The database vendors (like Oracle, MySQL) must implement the JDBC API.
- JDBC is based on the X/Open SQL Call Level Interface (CLI). JDBC 4.0 complies with the SQL 2003 standard.



Steps to write database code

1. Load the driver
2. Obtain connection
3. Create and execute statements
4. [Use result sets to navigate the results]
5. Close the connection

Load the driver

- `java.sql.DriverManager` class is used to get drivers and it's connection to the database.

- To register (Load) the driver with the application explicitly:

```
Class.forName("<driver class name>");
```

Or

```
DriverManager.registerDriver( new <driver class  
name>() );
```

Both of these, register the given driver with the `DriverManager`.. Static block of the `<Driver class>` calls `DriverManager.registerDriver()` method!

- With JDBC 4.0, this class also provides a mechanism, to automatically load the database drivers (provided, they are packaged in the specified way).

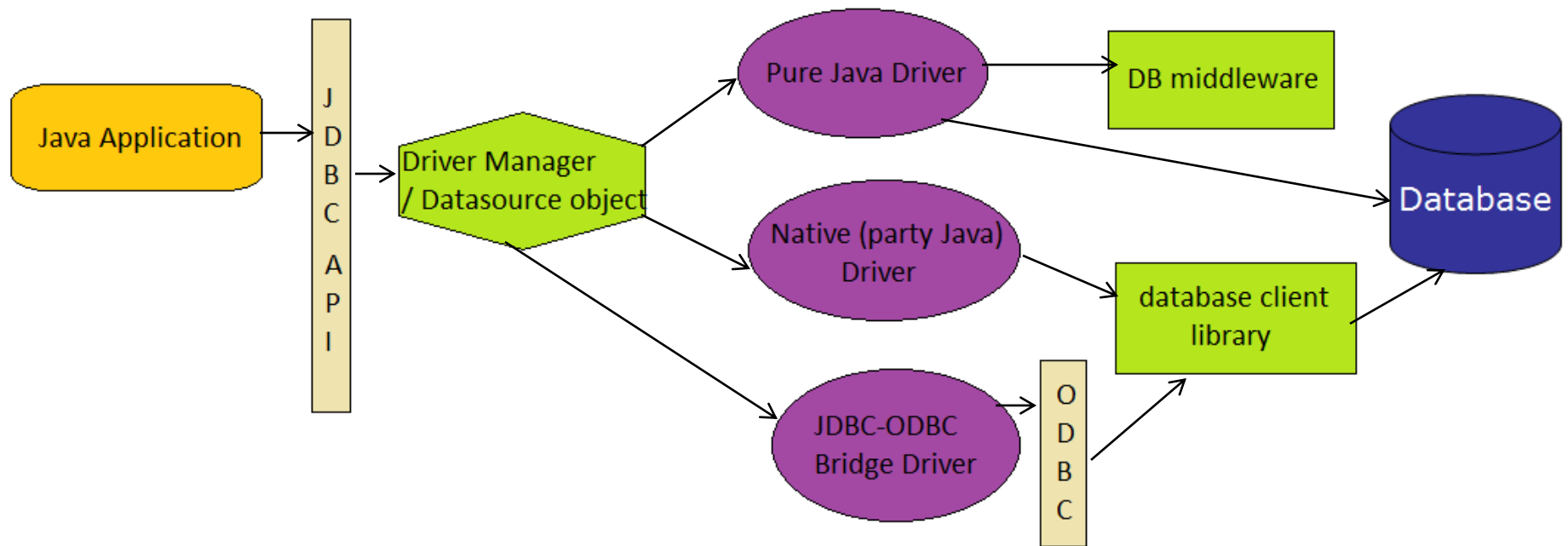
java.sql.Connection

- **Connection** is an interface.
- **getConnection()** method of **DriverManager** returns the object, that implements this interface.
- Connection class is used to do a variety of tasks
 - Obtain **Statement** object
 - To work with transactions
 - To get meta-data about the database
- For executing a static SQL statement, and returning the results it produces, an object that implements **Statement** interface is used.



Go through the Connection class API

JDBC Architecture

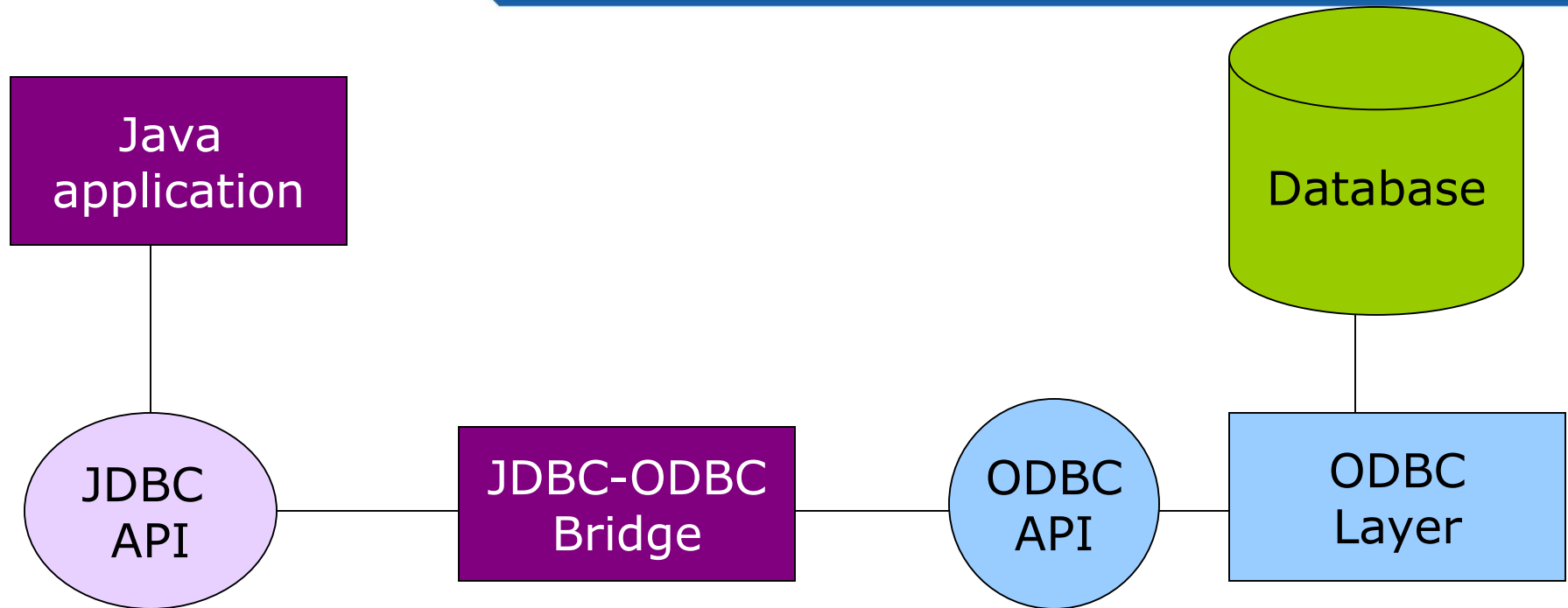


4 Ways to connect to database – through 4 types of driver

Driver Types

- JDBC driver are classes, that translate JDBC calls to either vendor-specific database calls or directly invoke database commands.
- Types of driver
 - ✓ Type 1- JDBC-ODBC Bridge
 - ✓ Type 2- Part Java, Part Native Driver
 - ✓ Type 3- Intermediate Database Access Server
 - ✓ Type 4- Pure Java Drivers

JDBC-ODBC Bridge

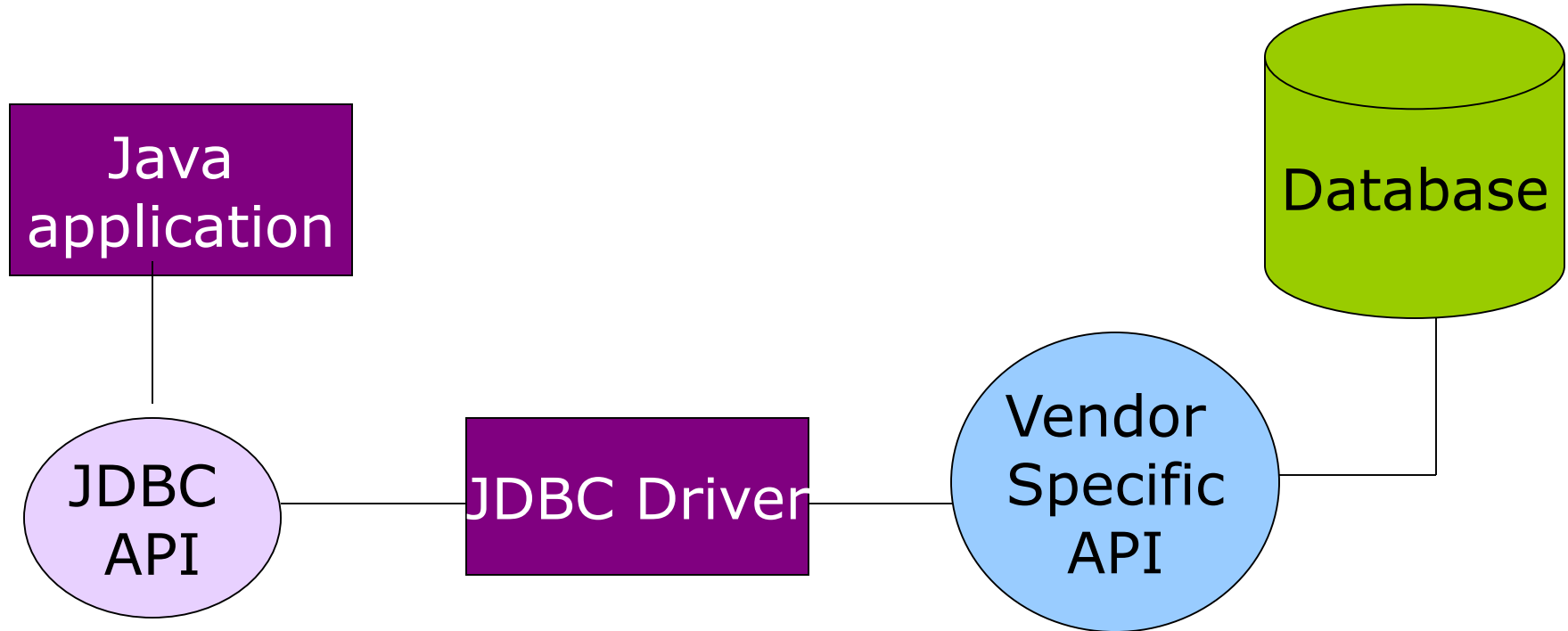


- They are driver classes that implement classes that implement the JDBC API as a mapping to ODBC (Open Database Connectivity) API.
- ODBC Microsoft's interface is used for accessing data in a heterogeneous database management system.
- ODBC binary code and in many cases, database client code -- must be loaded on each client machine, that uses a JDBC-ODBC Bridge

Disadvantages

- ODBC uses a C. The calls from Java to the native C code have a number of drawbacks; in terms of security, robustness and the automatic probability of applications.
- Multiple layers of indirection leads to inefficiency.

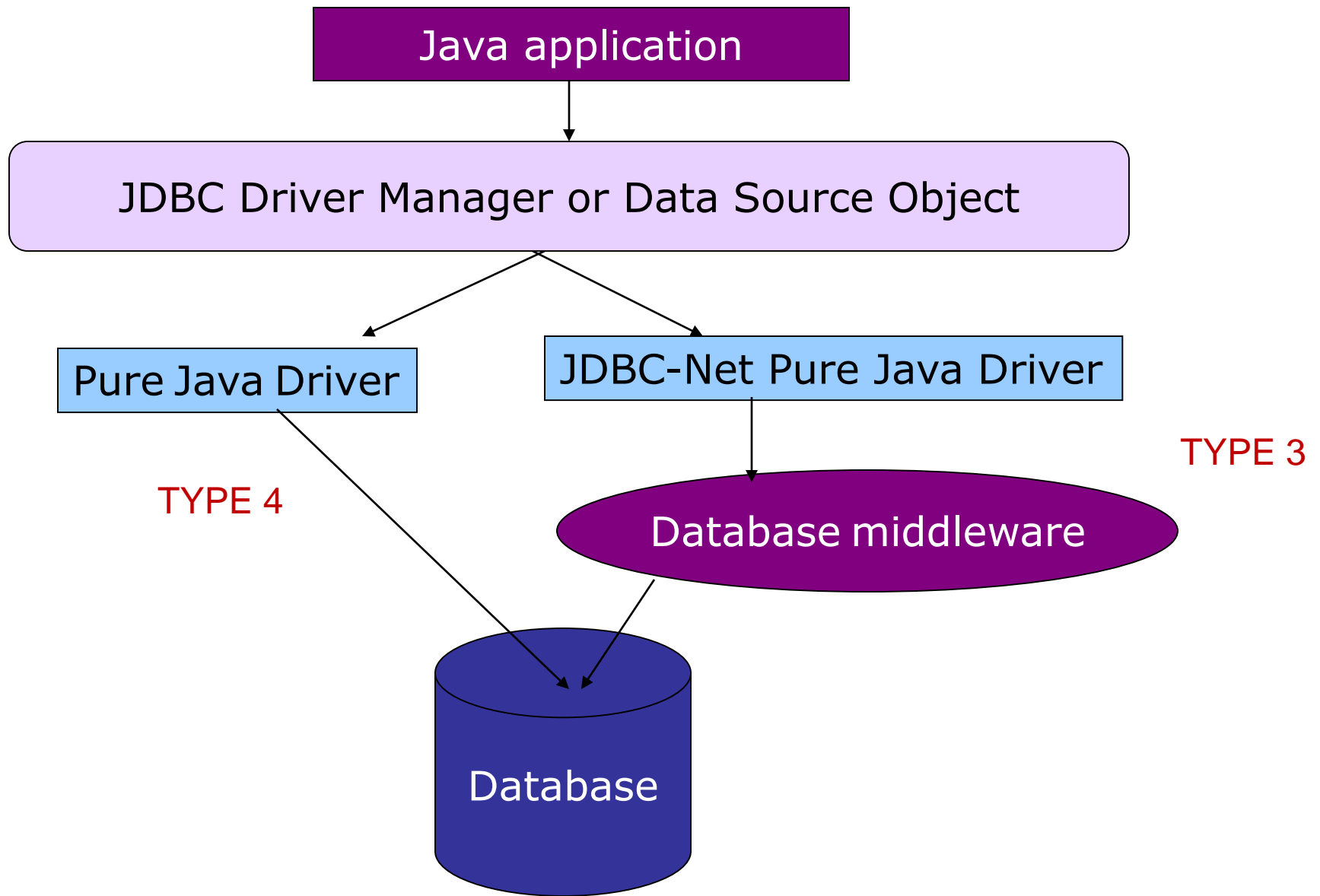
Type2- Part Java, Part Native Driver



Much like the bridge driver, this style of driver also requires some binary code to be loaded on each client machine. Therefore, this driver also experiences the same disadvantages. This driver is very rarely used nowadays.

Preferred Database Drivers

- Type 3 and Type 4 drivers are the preferred database drivers for Java.
- Each of these drivers are used in different architectural situations.
- Type 4 Drivers
 - Used in 2-tier architecture
 - Connected directly to the Database using Java application
 - Pure Java Driver
- Type 3
 - Three-tier
 - Connection to database happens through a middleware. The middleware provides connectivity to many different databases.
 - JDBC-Net pure Java Driver

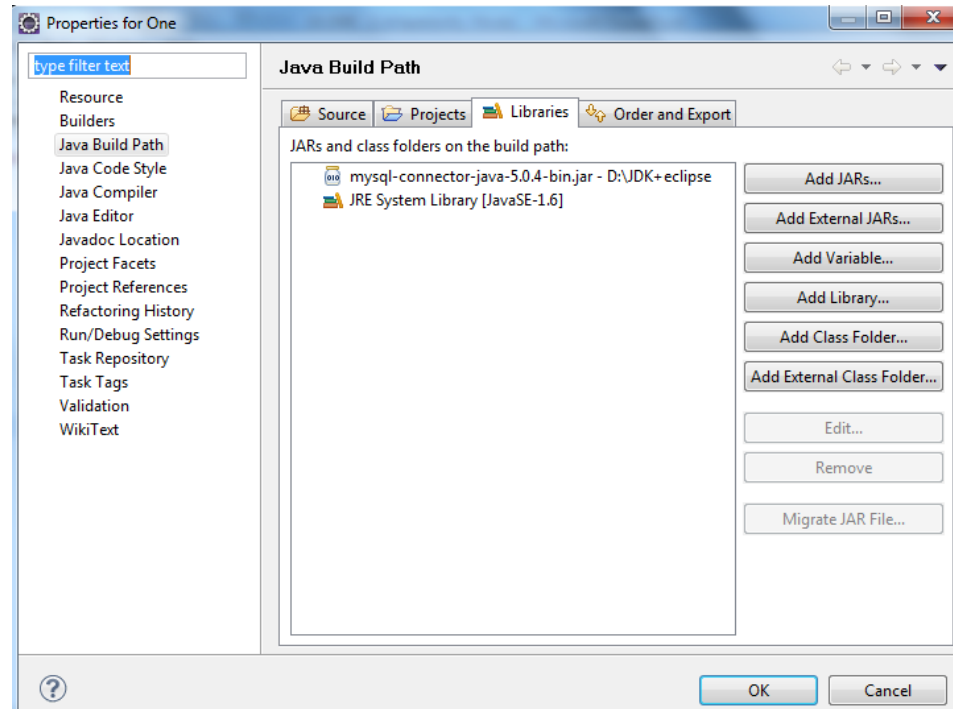


Connecting using type 4 driver

- There are two ways the code can be written; in order to obtain connectivity to database, using Type 4 driver:
 1. By specifying class explicitly in the code using `Class.forName()` or `DriverManager.registerDriver()`.
 2. By using JDBC 4.0 way, we can avoid hard coding the class name in the code.
- MySQL database using type 4 driver name is `com.mysql.jdbc.Driver`.
- Download Connector/J driver for `mysql-connector-java-5.0.4-bin.jar`

Setting classpath to a driver in eclipse

- Select your project, Go to Project menu and select Properties.
- Move to “Java Build Path” on the left panel. Click on “Add External JARs”, browse and locate the jar file for MySQL / Oracle.
- Click on Open. This will add the jar file to the Properties Box as shown in the diagram.



Example: Connecting to MySQL – older way

```
import java.sql.*;
import java.util.Properties;

public class Connect    {
    public static void main (String[] args) {
        Connection conn = null;
        try{
            String userName = "root";
            String password = "root";
            String url = "jdbc:oracle:thin:@127.0.0.1:1521:xe";
            Class.forName ("oracle.jdbc.driver.OracleDriver") ;
            Properties connectionProps = new Properties() ;
            connectionProps.put("user", userName) ;
            connectionProps.put("password", password) ;
```

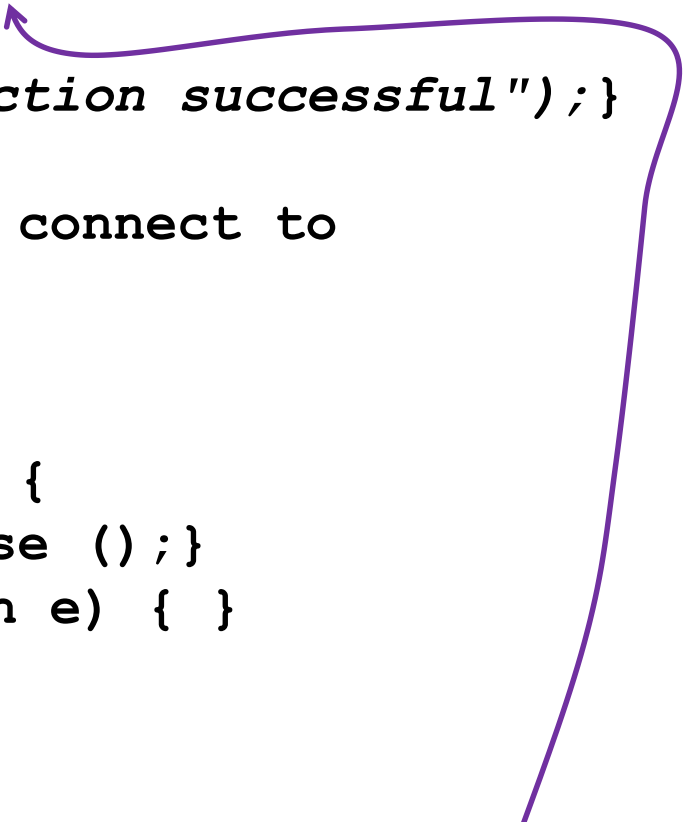


```

conn =
    DriverManager.getConnection(url,connectionProps);

System.out.println ("Database connection successful");}
catch (Exception e){
    System.err.println ("Failed to connect to
database" +e);
}
finally{
    if (conn != null)    {
        try    {    conn.close ();}
        catch (SQLException e) { }
    }
}
}
}

```




User name and password can also be sent as part of URL instead of properties. Find out the format in which it can be passed, if sent as a part of URL.

Result:

Database connection successful

JDBC 4.0 way to connect to database

- A JDBC 4-compliant Type 4 Driver JAR is identified, using the Java service provider mechanism. It contains a text file named “META-INF/services/java.sql.Driver”, which contains the name of the class(es) of the Drivers, which exist in that JAR.
- With JDBC 4.0, that uses the Java service provider mechanism, the applications no longer, applications no longer need to explicitly load JDBC drivers using `Class.forName()`.
- When the `getConnection()` is called, the DriverManager attempts to locate a suitable driver from the ones specified in the classpath. (If there are more than one, the first one is used.)

Code using JDBC 4.0 way

```
import java.sql.*;
import java.util.Properties;
public class Connect {
    public static void main (String[] args)
    {
        Connection conn = null;
        try {
            String userName = "root";
            String password = "root";
            String url = "jdbc:oracle:thin:@127.0.0.1:1521:xe";

            Properties connectionProps = new Properties();
            connectionProps.put("user", userName);
            connectionProps.put("password", password);
            conn = DriverManager.getConnection
                (url, connectionProps);
```

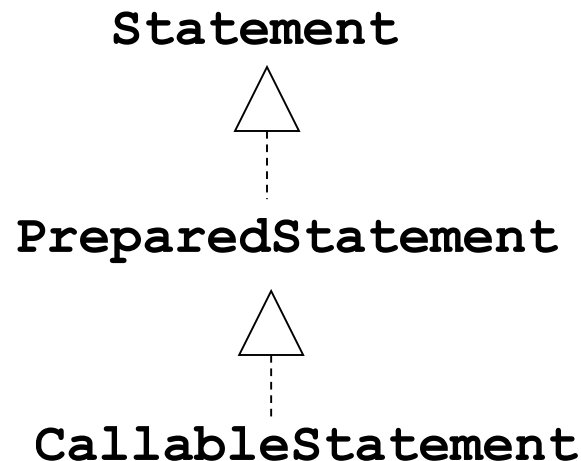
```

System.out.println ("Database connection
successful");
}
catch (SQLException e) {
System.err.println ("Failed to connect to database"
+e);
}
finally
{
    if (conn != null)    {
    try    {    conn.close ();}
    catch (SQLException e) { }
    }
}
}
}

```

Obtaining Statement

- `Connection` class methods to obtain `Statement` and its subclasses:
 - `Statement createStatement()` throws `SQLException`
 - `PreparedStatement prepareStatement(String sql)` throws `SQLException`
 - `CallableStatement prepareCall(String sql)` throws `SQLException`



SQLException

- Most methods in the `java.sql` and `javax.sql` packages throw **SQLException**.
- This object ,can be used to get message describing the error ,SQL state, error code specific to the database, and a reference to any chained exceptions

```
int getErrorCode()
```

```
int getErrorCode()
```

```
SQLException getNextException()
```

Statement

- The object of this type is used for executing a static SQL statement and returning its results.
- The result of the query statement is returned, in the form of **ResultSet** object .
- Only one **ResultSet** object per **Statement** object can be open at the one time.
- Therefore, reading data 2 or more **ResultSet** objects would require, obtaining **ResultSets** from different Statement objects.
- On calling execute methods on **Statement** object, the **ResultSet** object obtained from this **Statement** object (if there are any) is automatically closed.



Go through the Statement class API

ResultSet methods (default)

- **ResultSet** is an interface, representing table of data that is retrieved from a database
- It maintains a cursor initially pointing to the first row. The **next** method moves the cursor to the next row, until returns false when there are no rows to read.
- A default **ResultSet** object is not updatable and has a cursor that moves forward only.

boolean next() throws SQLException

void close() throws SQLException

XXX getXXX(int colIndex) throws SQLException

XXX getXXX(String colname) throws SQLException

where **XXX** is any primitive type, **String**, **java.sql.Date**
(subclass of **java.util.Date**), **Object**

colIndex begins from 1

colname as specified with the **SQL AS clause**

Code to insert and fetch records

```
Assuming a table in MYSQL named Student with regNo(int(11)),
name(varchar(45)), degree(name(varchar(45)) and
semester((int(11))
import java.sql.*;
import java.util.Properties;
public class Connect {
    public static void main (String[] args) {
        Connection conn = null;
        try
        {
            String userName = "root";
            String password = "root";
            String url = "jdbc:oracle:thin:@127.0.0.1:1521:xe";
            Properties props = new Properties();
            props.put("user", userName);
            props.put("password", password);
            conn = DriverManager.getConnection(url, props);
            Statement s= conn.createStatement();
```

```

s.executeUpdate("INSERT INTO STUDENT VALUES
(1, 'Rama', 'M.C.A.', 1)");
s.executeUpdate("INSERT INTO STUDENT VALUES
(2, 'Sita', 'B.Tech', 2)");
ResultSet rs=s.executeQuery("SELECT * FROM STUDENT");
System.out.println("ID      Name      Degree
Semester");
while (rs.next() ) {
System.out.println( rs.getInt(1) +"
"+rs.getString(2)+"      "+rs.getString(3)+"
"+rs.getInt(4));  }
} catch (SQLException e){
System.err.println ("Failed to connect to database" +e);
}
finally {
if (conn != null) {
try      { conn.close ();}catch (SQLException e) { }
} } } }

```

Exercise

- *Create a table called Training with the following fields,*
 - *Sap_ID*
 - *Employee_name*
 - *Stream*
 - *Percentage*
- *Enter some data into the table without Percentage.*
 1. *Write Java code using JDBC driver to display records.*
 2. *Each time the record is displayed , prompt the user to enter the Percentage and update the record.*

(1 hours)

Advanced ResultSet

- A default `ResultSet` object is not updatable and has a cursor that moves forward only.
- In order to have `ResultSet` which are scrollable and updatable, a different `createStatement()` method has to be used.
- `public Statement createStatement(int resSetType, int resSetConcurrency) throws SQLException`
- `resSetType` :
 - `ResultSet.TYPE_FORWARD_ONLY`
cursor may move only forward (default)
 - `ResultSet.TYPE_SCROLL_INSENSITIVE`
scrollable, but not sensitive to changes to the underlying data in the database, which happens outside the purview of this object.
 - `ResultSet.TYPE_SCROLL_SENSITIVE`
scrollable but sensitive to changes to the underlying data

- **resSetConcurrency** :

ResultSet.CONCUR_READ_ONLY

makes the result set read only

ResultSet.CONCUR_UPDATABLE

makes the result set updateable. Using this object, the rows can be inserted, updated, and deleted in the object itself, which automatically synchronizes with the database.

More ResultSet methods



*Go through various navigational methods of
ResultSet class*

Code using advanced ResultSet

```
import java.sql.*;
import java.util.Properties;
public class AdvRS {
    public static void main (String[] args)      {
        Connection conn = null;
        try
        {
            String userName = "root";
            String password = "root";
            String url = "jdbc:oracle:thin:@127.0.0.1:1521:xe";
            Properties props = new Properties();
            props.put("user", userName);
            props.put("password", password);
            conn = DriverManager.getConnection (url,props);

            Statement stmt =
            conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
            ResultSet.CONCUR_UPDATABLE);
```



```

ResultSet rs = stmt.executeQuery("SELECT * FROM
STUDENT");
System.out.println("Before...");
System.out.println("ID          Name          Degree
Semester");
while (rs.next() ) {
System.out.println( rs.getInt(1) +"
"+rs.getString(2)+"          "+rs.getString(3)+"
"+rs.getInt(4));
}
//inserting a new row
    rs.moveToInsertRow();
    rs.updateInt("RegNo", 3);
    rs.updateString("name", "Geeta");
    rs.updateString("degree", "B.E.");
    rs.updateInt("semester", 3);
    rs.insertRow();

```

```
//updating 2nd row - changing name to Seetha
```

```
rs.absolute(2);
```

```
rs.updateString(2,"Seetha");
```

```
rs.updateRow();
```

```
rs.beforeFirst();
```

```
System.out.println("After...");
```

```
System.out.println("ID      Name      Degree  
Semester");
```

```
while (rs.next() ) {
```

```
System.out.println( rs.getInt(1) +"
```

```
"+rs.getString(2)+"          "+rs.getString(3)+"
```

```
"+rs.getInt(4));  }}
```

```
catch (SQLException e) { System.err.println ("Failed to  
connect to database" +e);
```

```
}
```

```
finally {if (conn != null)      {
```

```
try {conn.close ();} catch (SQLException e) { }      }
```

```
}      }
```

Result of execution of the code

Before...

ID	Name	Degree	Semester
1	Rama	M.C.A.	1
2	Sita	B.Tech	2

After...

ID	Name	Degree	Semester
1	Rama	M.C.A.	1
2	Seetha	B.Tech	2
3	Geeta	B.E.	3

Exercise

- *Write a menu driven code to search, insert, update and delete from the Training table. Use advanced result set to do these.*

(2 hour)

PreparedStatement

- Interface that inherits from **Statement**
- If the same sql statement is executed many times, it is more efficient to use a prepared statement.
- It enables a SQL statement to contain parameters like functions. So, the same statement can be executed for different set of values.



*Go through the **PreparedStatement** API*

Inserting large objects in the database

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

public class InsertPhoto{
    public static void main(String[] args) {
        System.out.println("Insert Image Example!");
        String driverName = "jdbc:oracle:thin:@127.0.0.1:1521:xe";
        String userName = "root";
        String password = "root";
        String url = "jdbc:mysql://localhost/test";
        Connection con = null;
        try{
            Class.forName(driverName);
            con = DriverManager.getConnection(url,userName,password);
```

```
File imgfile = new File("D:\\image.jpg");
FileInputStream fin = new FileInputStream(imgfile);
PreparedStatement pre = con.prepareStatement("insert
into Photo values(?,?,?)");
pre.setInt(1,5);
pre.setString(2,"Durga");
pre.setBinaryStream(3,fin,(int)imgfile.length());
pre.executeUpdate();
System.out.println("Inserting Successfully!");
pre.close();
con.close();
}
catch (Exception e){
System.out.println(e.getMessage());}
}
```

Read object to file

- `Statement st = con.createStatement();`
- `ResultSet rs1 = st.executeQuery("select fileobj from photo where id = 6");` `InputStream is = null;`
- `Blob imageBlob = null;`
- `while(rs1.next())`
- `{`
- `imageBlob = rs1.getBlob(1);`
- `}`

- `//is = imageBlob.getBinaryStream(0, imageBlob.length());`
- `byte[] imageBytes = imageBlob.getBytes(1,
(int)imageBlob.length());`
- `FileOutputStream f = new FileOutputStream(new
File("C:\\scrap\\newFile.jpg"));`
- `f.write(imageBytes);`

Exercise

- *Update the Training table to include photos also.*
- *Write java code that allows uploading photos to the database, for existing records in the table.*

(1 hour)

PreparedStatement for batch updates

- Apart from the batch related methods, added by the **Statement** interface, **PreparedStatement** adds one more.
- **void addBatch() throws SQLException**
is used to add batch of commands
- But, since most of the batch operations are executed using same sql statement, it is usually used with **PreparedStatement**. Prepared Statement. Since it allows parameterized statements, they can be cached to give better performance.

Example: Batch updates

```
import java.sql.*;
public class BatchEx {
    public static void main (String[] args) throws
SQLException {
        Connection conn = null;
            String userName = "root";
            String password = "root";
            String url ="jdbc:mysql://localhost/test";
java.util.Calendar  c=java.util.Calendar.getInstance();
        conn =
DriverManager.getConnection(url,userName,password);
PreparedStatement stmt = conn.prepareStatement("INSERT
INTO BookIssue VALUES (?, ?, ?, ?)");
stmt.setInt(1,15);
stmt.setInt(2,1);
c.clear();    c.set(2011,9,25);
stmt.setDate(3, new
java.sql.Date((c.getTime()).getTime()));
```

```
c.set(2011,9,27);  
stmt.setDate(4,new  
java.sql.Date((c.getTime()).getTime()));
```

```
stmt.addBatch();  
stmt.setInt(1,16);  
stmt.setInt(2,2);  
c.clear();  
c.set(2011,10,12);  
stmt.setDate(3,new  
java.sql.Date((c.getTime()).getTime()));  
c.set(2011,10,27);  
stmt.setDate(4,new  
java.sql.Date((c.getTime()).getTime()));  
stmt.addBatch();  
stmt.executeBatch();  
conn.close();  
}}
```

CallableStatement

- Interface that inherits from **PreparedStatement** is used to execute stored-procedure.
- **Connection.prepareCall(String s)** is used to create a **CallableStatement**.
- The string parameter is of the form
{[?=call** <procedure-name>[<arg1>,<arg2>, ...]}**
- The arguments IN parameter values are set using the set methods, inherited from **PreparedStatement**.
- All **OUT** parameters must be registered before a stored procedure is executed.

**void registerOutParameter(int parameterIndex,
int sqlType) throws SQLException**

- **java.sql.Types** has all the parameters that can be sent as **sqlType**. (Please refer to <docfolder>\docs\api\java\sql\Types.html)

Java Code to call the stored procedure

```
Create PROCEDURE GETSTUD(IN id1 INT,OUT nm VARCHAR(45))
BEGIN
    SELECT SNAME INTO nm FROM STUDENT WHERE sid=id1;
END
```

```
import java.sql.*;
public class CallSPMySQL {
    public static void main(String[] args) {

        String userName = "root";
        String password = "root";
        String url = "jdbc:mysql://localhost/test";
        Connection conn = null;
        try{

            conn =
            DriverManager.getConnection(url,userName,password) ;
            CallableStatement c=conn.prepareCall("{call
            GETSTUD(?,?) }");
```

```

String name=null;
c.setInt(1,2);
c.registerOutParameter(2,java.sql.Types.VARCHAR);
c.executeUpdate();
name=c.getString(2);

System.out.println("Name retrieved: "+ name);
}
catch (SQLException e) {
System.err.println ("Failed to connect to database" +e);
}

        finally {
            if (conn != null)
                try { conn.close ();} catch
(SQLException e) { }
        }
}
}

```


Get ResultSet from stored procedure

Using `execute()`, `getResultSet()` and `getMoreResults()` methods of the `Statement`, multiple results can be retrieved from stored procedure.

```
CREATE PROCEDURE AllStudents()  
BEGIN  
Select * from Student;  
END
```

```
import java.sql.*;  
public class CallableSQLRS {  
public static void main(String[] args) {  
  
String userName = "root";  
    String password = "root";  
    String url = "jdbc:mysql://localhost/test";  
Connection conn = null;  
try{  
conn =  
DriverManager.getConnection(url,userName,password) ;
```

```

CallableStatement c=conn.prepareCall("{call
AllStudents()}");
boolean res = c.execute();
while (res) {
    ResultSet rs = c.getResultSet();
    System.out.println("ID          Name          Degree
Semester");
    while (rs.next() ) {
        System.out.printf( "%2s %10s %10s
%7s\n",rs.getInt(1),rs.getString(2),rs.getString(3),rs.g
etInt(4));
    }
    res = c.getMoreResults();
}
}
catch (SQLException e) { System.err.println(e);}
finally { if (conn != null)
try {conn.close ();} catch (SQLException e) { }
}    }}

```

Exercise

- *Create a table called Account with AccNo, AccName, AccType and Balance. AccNo is the primary key.*
- *Write a stored procedure that returns the new AccNo which is computed as last entered AccNo +1. If the record is new, then AccNo is 1111111.*
- *Write java program , to allows insertion into the Account table.*

(1 hour)

Calling a function

- `cStmt = con`
- `.prepareCall("select emp_count(?) from dual");`
- `cStmt.setDouble(1, 10);`
- `ResultSet rs = cStmt.executeQuery();`
- **`while (rs.next()) {`**
- `System.out.println(rs.getInt(1));`
- `}`

Summary

- JDBC 4.0 is an API, that provides standard for connectivity to a variety of data sources like SQL databases, spreadsheets and flat.
- The following are the steps to write database code: Load the driver, Obtain connection, Create and execute statements, Use result sets to navigate the results and Close the connection.
- JDBC driver are classes, that translate JDBC calls to either vendor-specific database calls or directly invoke database commands.
- Type 1- JDBC-ODBC Bridge, Type 2- Part Java, Part Native Driver, Type 3- Intermediate Database Access Server and Type 4- Pure Java Drivers are the four types of driver.
- Most methods in the `java.sql` and `javax.sql` packages throw `SQLException`.