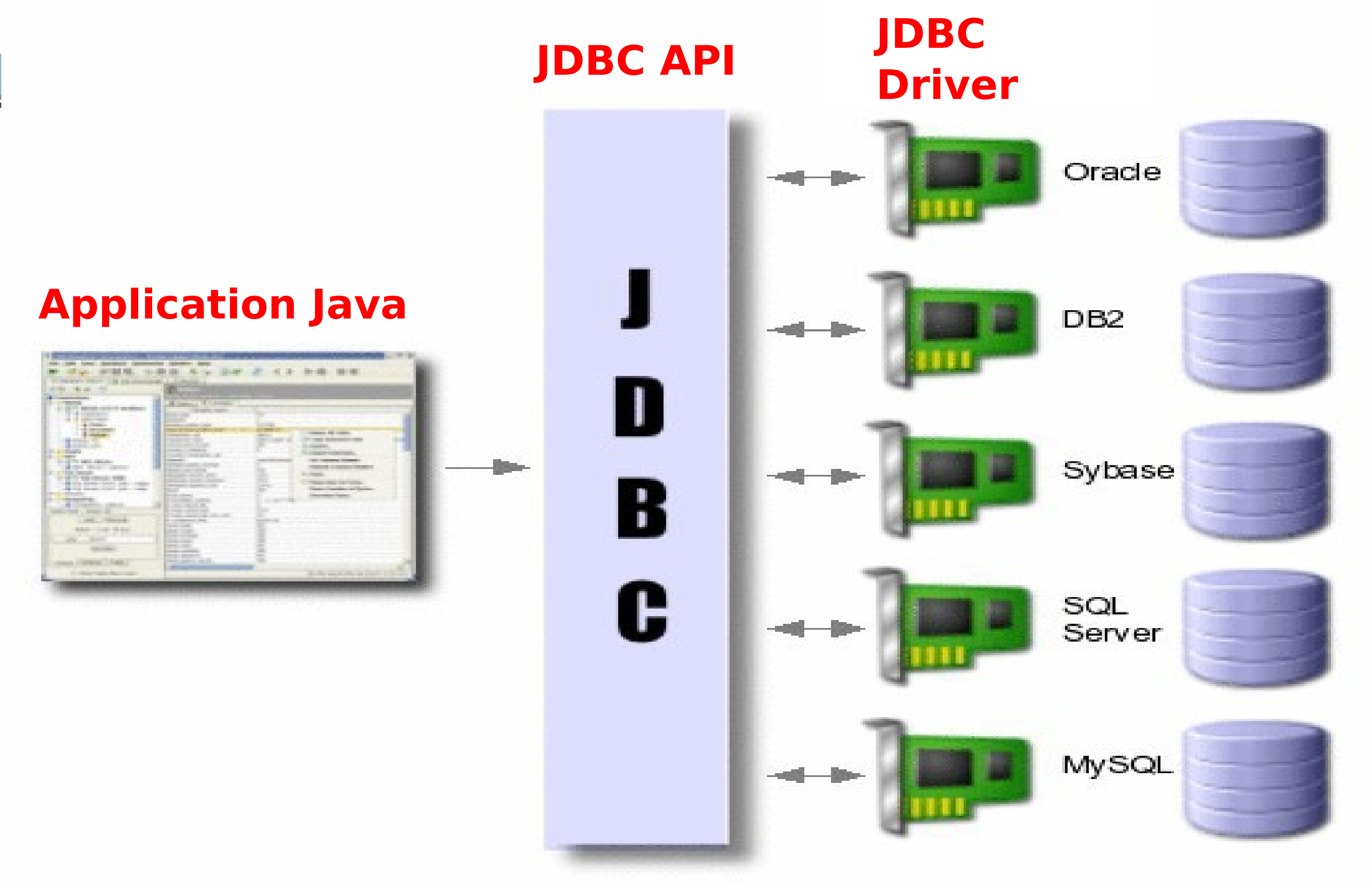
420-P33-SU
OBJECT ORIENTED PROGRAMMING II

Agenda

- Overview of JDBC Technology
- JDBC Drivers
- Data Retrieval (ResultSet)
- Using Statements and PreparedStatements
- The concept of transactions


JDBC API
JDBC Driver
Application Java
JDBC
Oracle
DB2
Sybase
SQL Server
MySQL

JDBC Technology

- Standard database access library
- 🗣 Allows for detail abstraction from DB to our application.
- The API :
  - One way to connect to the database.
  - Method for creating parameterized queries.
  - Way to determine the structure of the query's result :
    - Number of columns
    - Metadata

JDBC Technology

- JDBC is a set of classes in the java.sql package
- Defines a set of interfaces that are implemented by drivers (editors)
- Java.sql contains:
  - DriverManager,
  - Connection,
  - ResultSet,
  - DatabaseMetaData,
  - ResultSetMetaData,
  - PreparedStatement,
  - CallableStatement

JDBC Driver

- Specific implementation of JDBC interfaces according to the publisher's DB
- Each DB editor has its own JDBC driver

Steps for using JDBC

1. Load the driver.
2. Define the url connection.
3. Establish the connection.
4. Create the Statement (or PreparedStatement) object.
5. Run the query.
6. Read the results.
7. Close the connection.

Loading the driver

Manual loading of the target DB driver and registration with DriverManager

Example:

```
try{
Class.forName(“com.mysql.cj.jcbc.Driver”);
//Class.forName(“connect.microsoft.MicrosoftD
river”);
} catch { ClassNotFoundException (e) {
System.out.println(“Driver Error: ” + e);
}
```

Defining the URL connection

```
String
  mysqlURL=‘jdbc:mysql://localhost:3306/auto?
  serverTimezone=UTC’;
```

Establishing the connection

DriverManager is responsible for selecting the DB and creating the connection

- **Example**

```
String identifier = "HR";
String password = "secret";
Connection connection =
DriverManager.getConnection(mysqlURL, identifier,
  connection);
```

Creation of the Statement Object

- Creating a Statement object from the connection object

- **Example**

```
Statement statement = connection.createStatement();
```

- The <u>Statement</u> object is responsible for sending SQL commands to the database and recuperating the results of the commands.

Execution of the request

- In query mode:

```
String query ="SELECT department_name from departments";
 ResultSet resultSet = statement.executeQuery(query);
```

- In modification mode (CUD), use the following method:

executeUpdate(query)with a parameter: a string that uses UPDATE, INSERT, or DELETE..

```
Int i = statement.executeUpdate(query);
```

Evaluation of the result

Example:

```
while(resultSet.next()) {
  System.out.println(resultSet.getString(1);
}
```

- We need to loop through the resultSet to retrieve the results of the SQL execution.
- Column 1 has the index **<u>1</u>** (be careful that it is not 0)
- Use getXXX () methods to retrieve data by column index
- Note: we can access the metadata (to have for example the name of the columns)
- Tip: consider putting the result in a Collection !!

Closing the Connection

```
connection.close();


//Stop today
```

Prepared Statements

- **Using PreparedStatements**

- Use parametric statements

- Create a standard statement that is sent for compilation at the DB level before being used

- With each use, we replace the (marked) parameters using the setXXX () methods

- Faster than the Statement because it is prepared before use

Prepared Statements (precompiled)

- Use the « **<u>?</u>** » symbol as a parameter marker
- Markers are numbered, starting with 1 on the left.
- execute() methods after variables are bound (setXXX), that is :
  - execute()
  - executeQuery()
  - executeUpdate()
- These methods take no parameters.

Prepared Statements (precompiled)

Example

```
stmt = conn.prepareStatement("select first_name,
  last_name, salary, commission_pct from
  employees  where department_id= ?");

//set department code
stmt.setInt(1,80);
```

Prepared Statements - Methods

- setXXX: sets the specified parameter (with ?) in the SQL statement to the indicated value.
- clearParameters(): resets all the statements parameter values

Recuperating MetaData

- Once you have the ResultSet or the Connection object, you can retrieve :
  - Metadata concerning the DB or the request.
- Metadata is information regarding :
  - The Data that we recovered.
  - The DB that we are using.

- Example:

```
ResultSet result =stmt.executeQuery();
ResultSetMetaData meta = result.getMetaData();
ResultSetMetaData dbmeta = connection.getMetaData();
```

Example of MetaData :Column name and type

```
//managing metaData

ResultSetMetaData resultMetaData = result.getMetaData();

int nbmColumn = resultMetaData.getColumnCount();

for (int i=1;i<= nbmColumn; i++){
  String nameColumn = resultMetaData.getColumnName(i);
  String typeColumn = resultMetaData.getColumnTypeName(i);
  System.out.println("Column number: "+i+" has the
  name:  "+nameColumn+" with a type of: "+typeColumn);
}
```

Transactions

- Commit() calls slow down processing
- We make a commit for the whole transaction: need to set AutoCommit to **false**
- Gives us the opportunity to rollback in case of failure (Exception !!)

Transactional JDBC methods

- **setAutoCommit**(): if true, every statement that is executed is followed by an immediate commit.
- **commit**(): only works when setAutoCommit(false). Commits operations since the last commit, rollback, or login.
- **rollback**(): only works when setAutoCommit(false). Cancel all operations performed but not committed.

Example

```
Connection connection = null;
 try {

connection =  DriverManager.getConnection(
"jdbc:oracle:thin:@192.168.0.15
:1521:ORCL","hr","hr");

connection.setAutoCommit(false);

PreparedStatement updateQty =
connection.prepareStatement("UPDATE employees SET
salary = ? WHERE departement_id = ? ");

updateQty.setInt(1,qty);
updateQty.setInt(2,itemCode);
iRecordsUpdate +=
updateQty.executeUpdate();

connection.commit();

}
```

Example (cont.)

```
} catch(SQLException e)
{  System.out.println("" + e);
try {  connection.rollback();
} catch(SQLException sqleRollback) {
System.out.println("" + sqleRollback);
}
}
finally {
try {
connection.close();
}
catch(SQLException sqleClose) {
System.out.println("" + sqleClose);
}
```