JDBC

Topics

1) Intro to JDBC
2) Establishing a Connection
3) Loading Drivers
4) Connection Object
5) Statement
6) PreparedStatement
7) ResultSet
8) Transactions
9) Commit/Rollback

Intro to JDBC

- Java Database Connectivity
- Standard framework for handling tabular/relational data
- Located in java.sql package

Establishing a Connection

- The first thing to do is establish a connection with the DBMS
- This involves two steps:
  - Loading the driver
  - Making the connection

Loading Drivers

- Loading a driver is very simple
- One line of code

```
Class.forName ("oracle.jdbc.OracleDriver");
Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
Class.forName("com.mysql.jdbc.Driver").newInstance();
```

- You also need the respective JAR file in your build
- Drivers can be easily downloaded online

Connection Object

- To make the driver connect to the database:
- One line of code

```
Connection conn = DriverManager.getConnection(url, user, password);
```

- What's my URL?
  - Every vendor is different

```
String url = "jdbc:oracle:thin:@localhost:1521:xe";
String url = "jdbc:sqlserver://localhost:1433;databaseName=MyDatabase";
String url = "jdbc:mysql://localhost:3306/MyDatabase";
```

Statement

- Once connected, you can execute SQL statements
- A Statement object wraps and executes SQL (including DDL, DML, etc.)

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

- For a SELECT statement, the method to use is executeQuery

```
stmt.executeQuery("SELECT * FROM TRAINEES");
```

- To INSERT or UPDATE, use executeUpdate

```
stmt.executeUpdate("UPDATE TRAINEES SET RANK = 'EXPERT' WHERE USER_ID = '5'");
```

- Review other methods of the Statement class

PreparedStatement

- Precompiled SQL statement
- Reduce execution time for repetitious SQL statements
- Parameterized inputs using '?' placeholders
  - Easier to read
- Database reserved characters automatically escaped
  - SQL Injection prevention

```
PreparedStatement updateEmp= conn.prepareStatement("UPDATE student SET fees = ? WHERE name LIKE ?");
updateEmp.setInt(1, 750);
updateEmp.setString(2, "John Doe");
updateEmp.executeUpdate();
```

- pstmt.method(Placeholder position, value to insert)

ResultSet

- Object that wraps the rows returned by the query
- Return type of a query

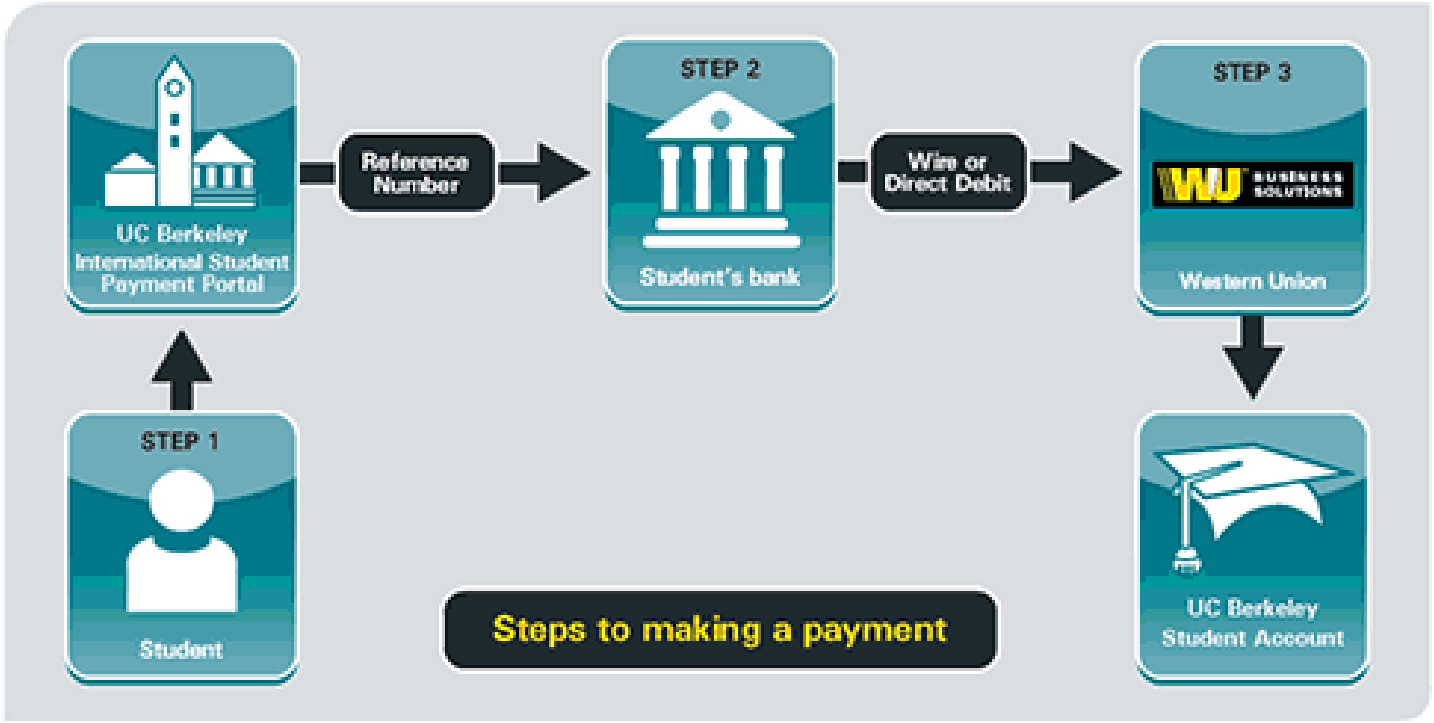
```
ResultSet rs = stmt.executeQuery("SELECT * FROM TRAINEES");
```

- Loop over the ResultSet to inspect each row
- Select each column by “column_name” or column_index
- At end of loop, ResultSet points to the next row

```
while(rs.next()){
    System.out.println(
                  rs.getString("name")
        + "\t" + rs.getInt("age")
        + "\t" + rs.getFloat("fees")
        + "\t" + rs.getDate("date_of_hire"));
}
```

Transactions

- ACID: Atomic, Consistent, Isolation, Durable



Transactions

- JDBC is in auto-commit mode by default
- Each Statement is immediately committed
- Disable auto-commit mode to execute Statements as a unit
- Re-enable auto-commit mode at the end of transaction

```
conn.setAutoCommit(false);

String deductFromStudent = "UPDATE STUDENT SET BALANCE = BALANCE - ? WHERE STUDENT_ID = ?";
String postPayment = "UPDATE ACCOUNT SET BALANCE = BALANCE + ? WHERE STUDENT_ID = ?";

PreparedStatement pstmt = conn.prepareStatement(deductFromStudent);
pstmt.executeUpdate();

pstmt = conn.prepareStatement(postPayment);
pstmt.executeUpdate();

conn.commit();
conn.setAutoCommit(true);
```

Commit/Rollback

- To make changes permanent, use commit( ) method
- To return to the previous state, use rollback( ) method

```
try{
    conn.setAutoCommit(false);

    String deductFromStudent = "UPDATE STUDENT SET BALANCE = BALANCE - ? WHERE STUDENT_ID = ?";
    String postPayment = "UPDATE ACCOUNT SET BALANCE = BALANCE + ? WHERE STUDENT_ID = ?";

    PreparedStatement pstmt = conn.prepareStatement(deductFromStudent);
    pstmt.executeUpdate();

    pstmt = conn.prepareStatement(postPayment);
    pstmt.executeUpdate();

    conn.commit();

}catch(Exception e){
    conn.rollback();
}finally{
    conn.setAutoCommit(true);
}
```

Just like Files…

- Always close resources after using them
- All JDBC resources have a close( ) method

```
statement.close();
preparedStatement.close();
connection.close();
```

Review

1) Intro to JDBC
2) Establishing a Connection
3) Loading Drivers
4) Connection Object
5) Statement
6) PreparedStatement
7) ResultSet
8) Transactions
9) Commit/Rollback

Assignment

- Create a Connection where the database URL, username, and password is retrieved from a Properties file.
- Create a table with at least 3 columns within the Java application.
- Insert test data into the table.
- Query the test data using Statement and PreparedStatement.
- Study CallableStatement for calling stored procedures in your database (PL/SQL, T-SQL, etc.)