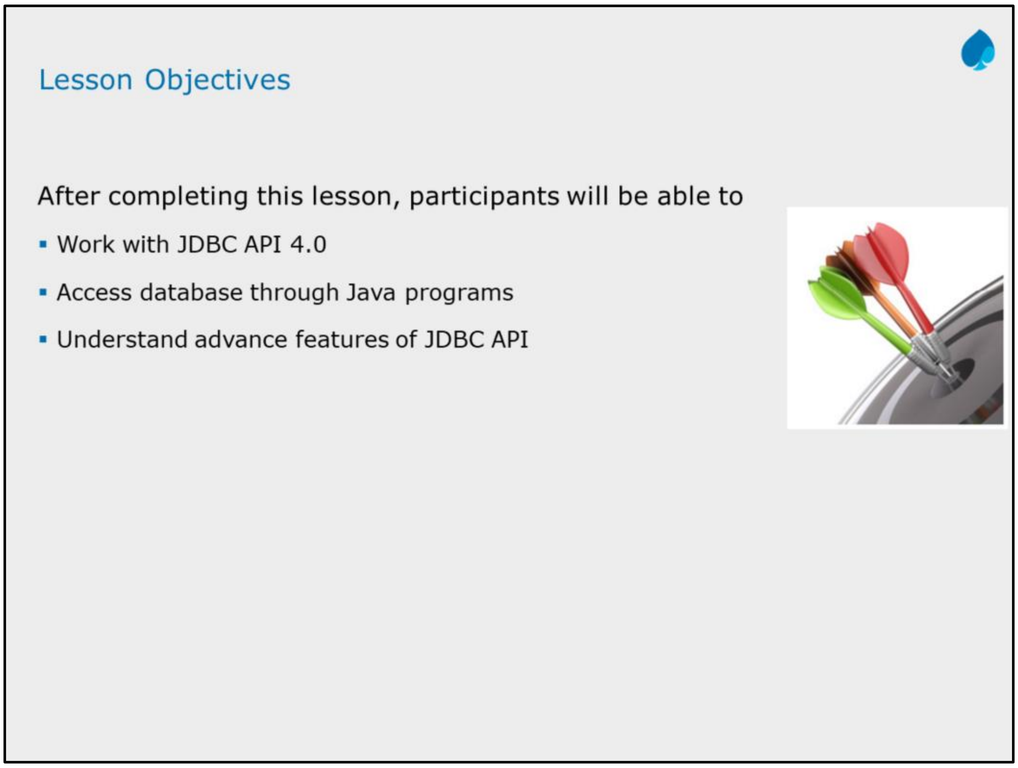
JDBC

Lesson 2 : Getting Started With JDBC

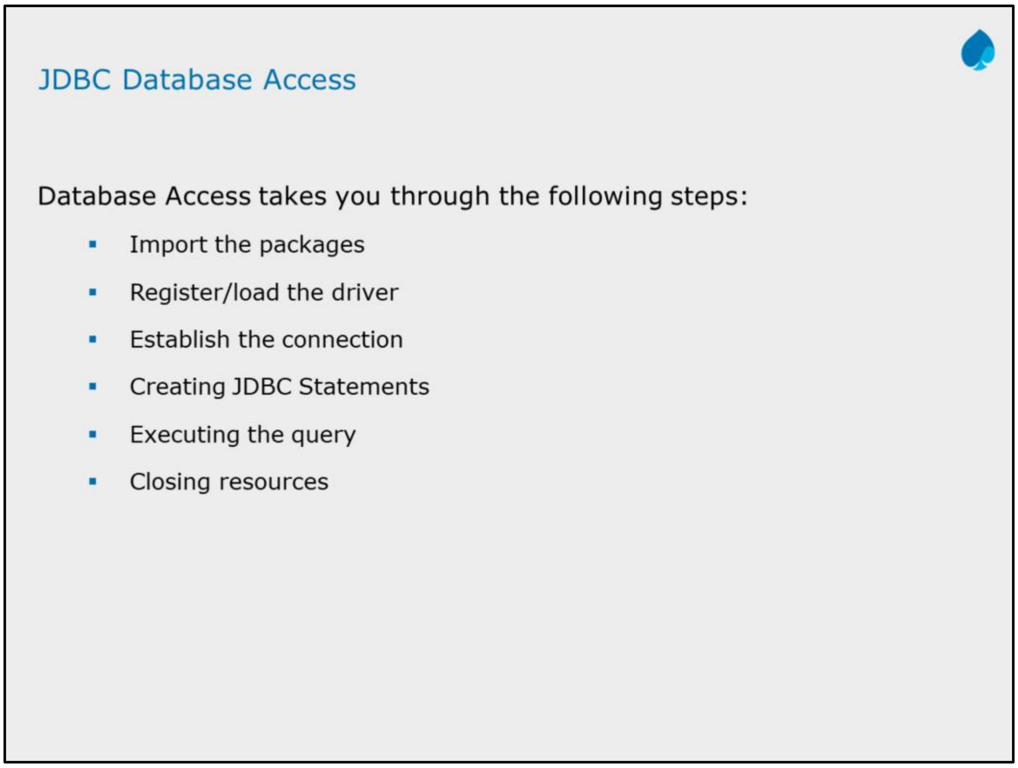
Capgemini



Lesson Objectives

After completing this lesson, participants will be able to

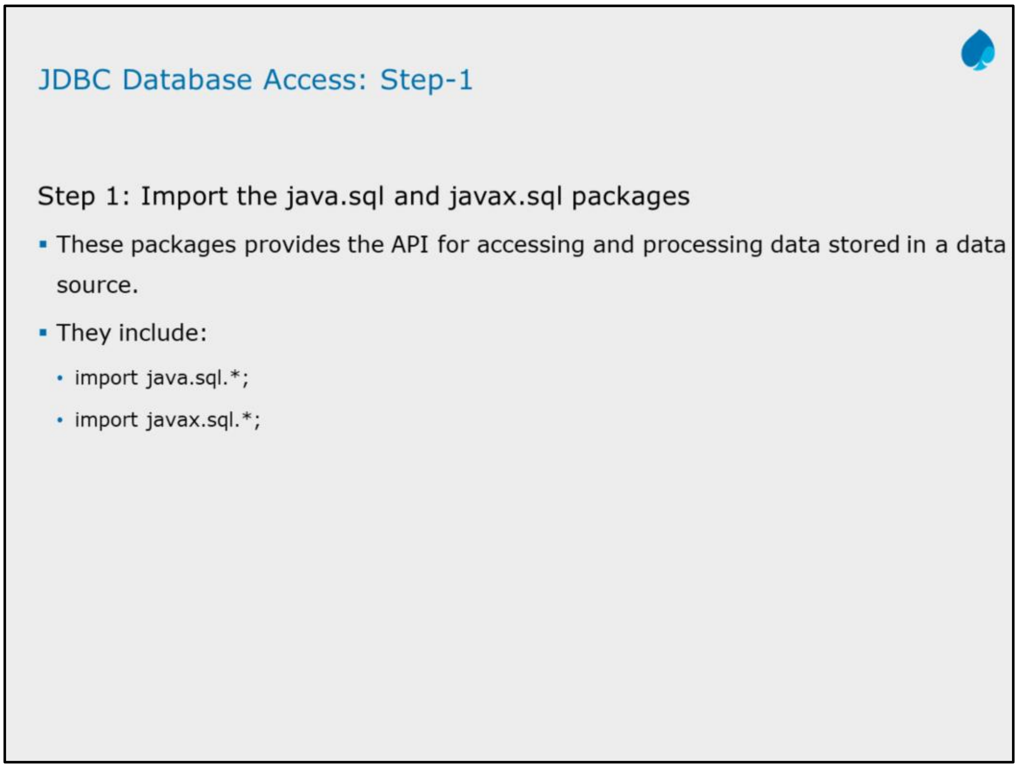
- Work with JDBC API 4.0
- Access database through Java programs
- Understand advance features of JDBC API



JDBC Database Access

Database Access takes you through the following steps:

- Import the packages
- Register/load the driver
- Establish the connection
- Creating JDBC Statements
- Executing the query
- Closing resources

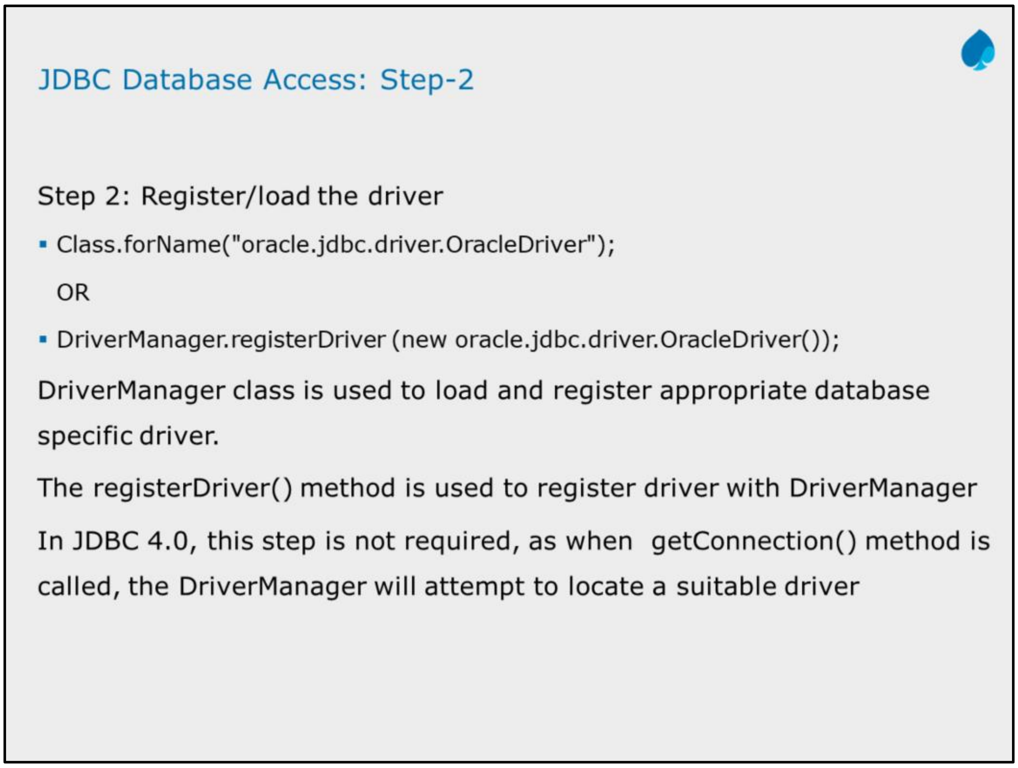


JDBC Database Access: Step-1

Step 1: Import the java.sql and javax.sql packages

- These packages provides the API for accessing and processing data stored in a data source.
- They include:
  - import java.sql.*;
  - import javax.sql.*;

Import Packages:
The first step in accessing the data from database using JDBC APIs is importing the packages java.sql and javax.sql.
These packages provides the set of APIs which are used in accessing the database like Connection, Statement, and so on.



JDBC Database Access: Step-2

Step 2: Register/load the driver

- Class.forName("oracle.jdbc.driver.OracleDriver");

  OR

- DriverManager.registerDriver (new oracle.jdbc.driver.OracleDriver());

DriverManager class is used to load and register appropriate database specific driver.

The registerDriver() method is used to register driver with DriverManager

In JDBC 4.0, this step is not required, as when getConnection() method is called, the DriverManager will attempt to locate a suitable driver

Register Driver:

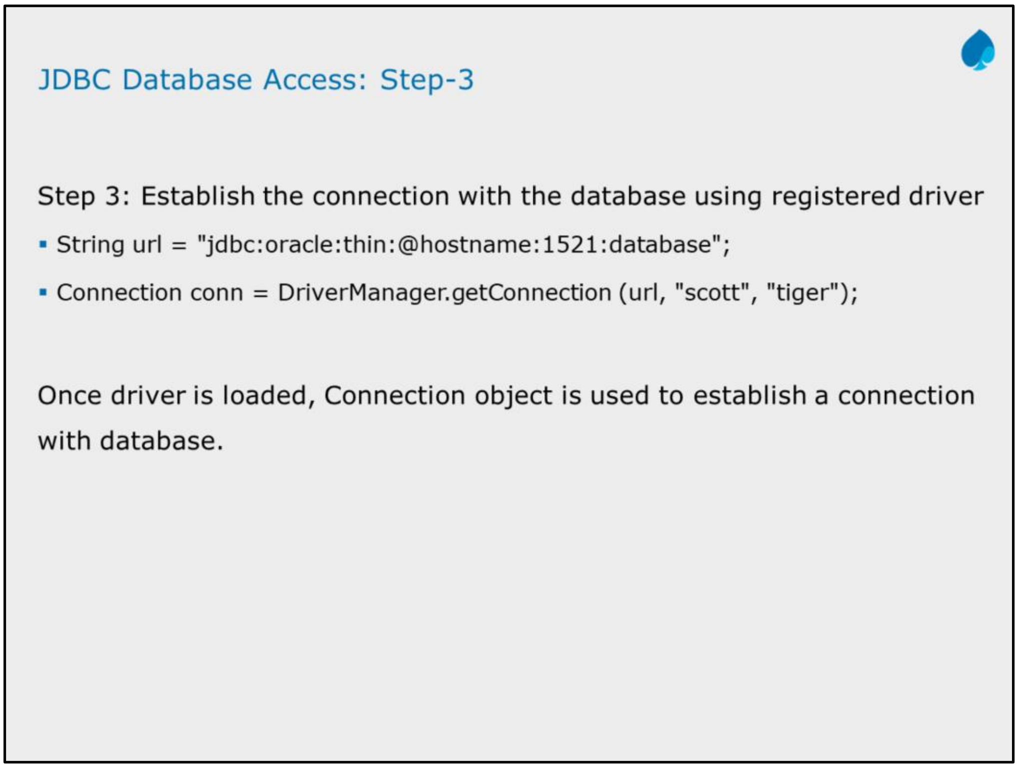
The second step is to register/load the driver for the respective database. There are two ways of loading the driver:

By using Class.forName() method

By calling DriverManager's registerDriver() method

Using Class.forName() you can load any class while DriverManger.registerDriver() is specific to JDBC driver class.

Note: In JDBC 4.0, we don't need the Class.forName() line. We can simply call getConnection() to get the database connection.



JDBC Database Access: Step-3

Step 3: Establish the connection with the database using registered driver

- String url = "jdbc:oracle:thin:@hostname:1521:database";
- Connection conn = DriverManager.getConnection (url, "scott", "tiger");

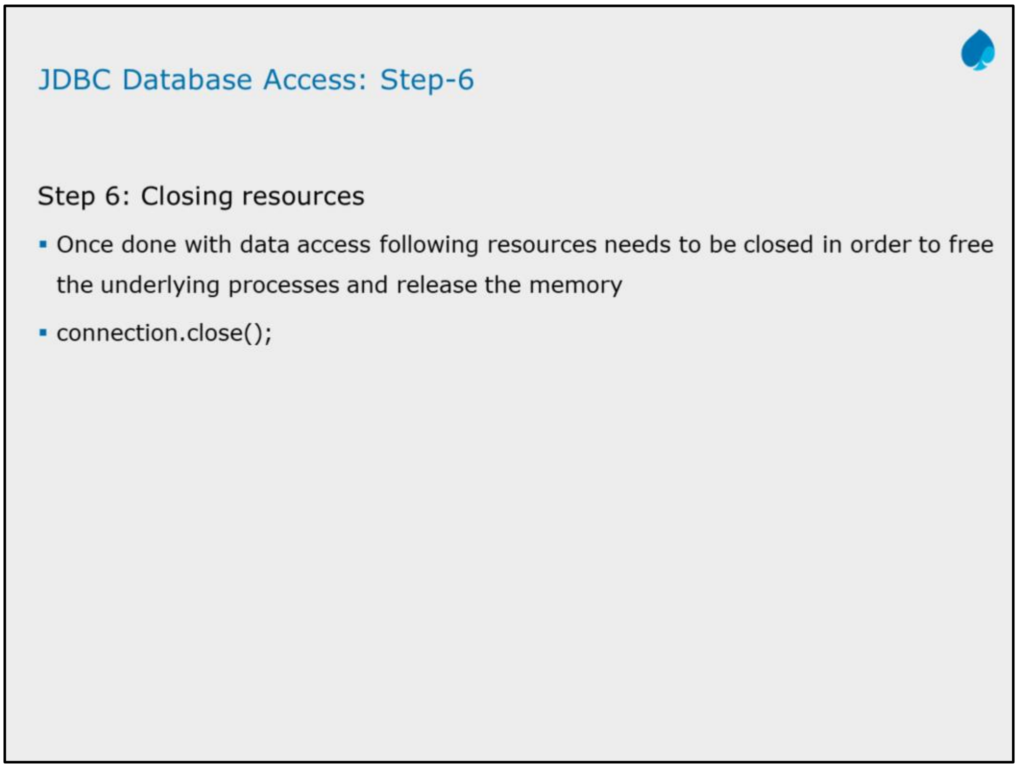
Once driver is loaded, Connection object is used to establish a connection with database.

Establish the Connection:

The third step is to establish a connection with the database. There is a method DriverManager.getConnection() which returns Connection object.

This method take three arguments,

URL: This contains the information about host on which database server, database name, and the port used for communication.
Username: database userid
Password: database password

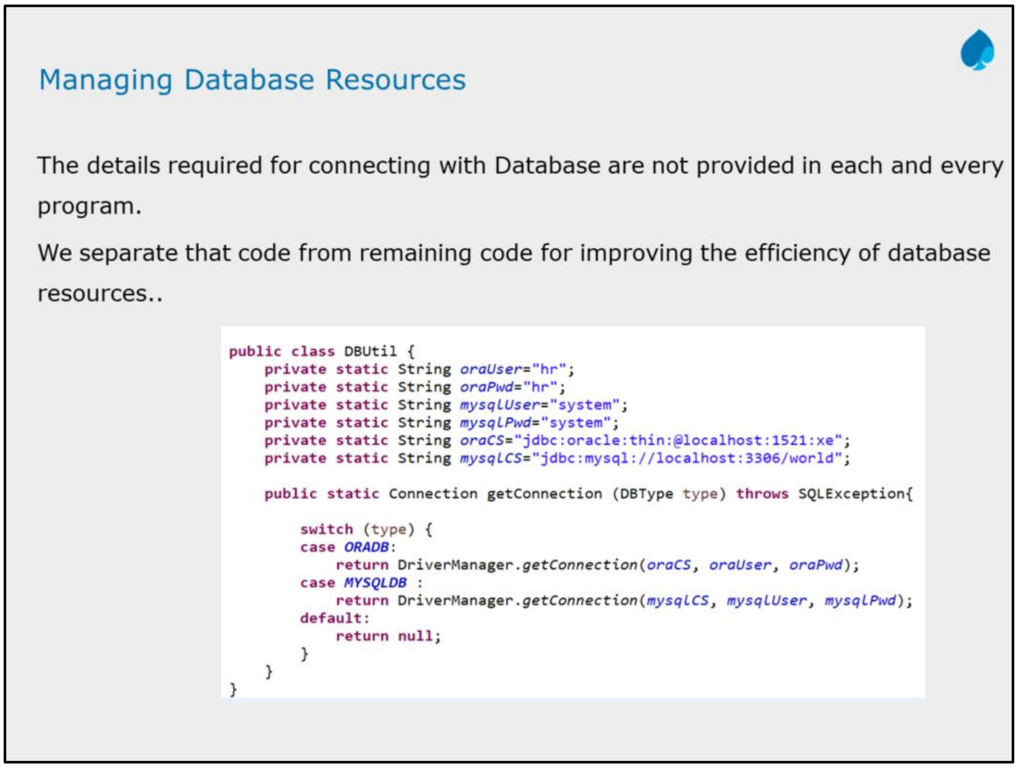


JDBC Database Access: Step-6

Step 6: Closing resources

- Once done with data access following resources needs to be closed in order to free the underlying processes and release the memory
- connection.close();

Closing resources:
The last step in database access is closing used resources in program. Closing statements and result sets is required in order to free the underlying processes and memory.
Connection object is also required to be closed. Failure in closing connection object results in database server running out of connections.
All of these interfaces provides close() method, which is used to close the respective resource.



Managing Database Resources

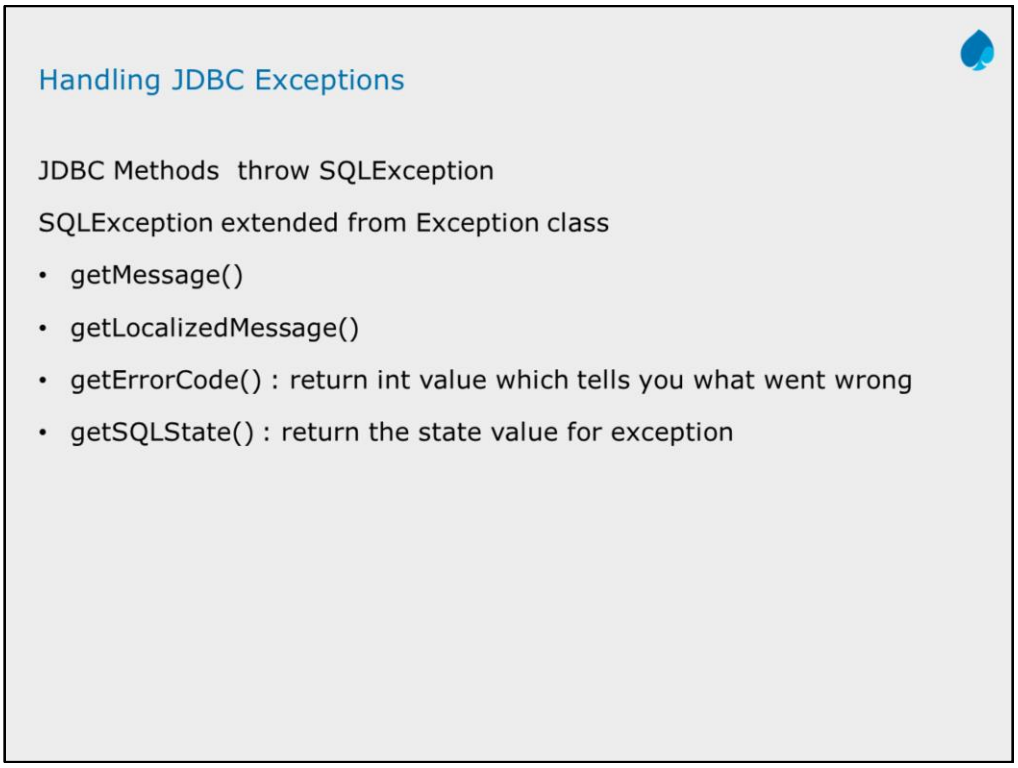
The details required for connecting with Database are not provided in each and every program.

We separate that code from remaining code for improving the efficiency of database resources..

```
public class DBUtil {
    private static String oraUser="hr";
    private static String oraPwd="hr";
    private static String mysqlUser="system";
    private static String mysqlPwd="system";
    private static String oraCS="jdbc:oracle:thin:@localhost:1521:xe";
    private static String mysqlCS="jdbc:mysql://localhost:3306/world";

    public static Connection getConnection (DBType type) throws SQLException{

        switch (type) {
        case ORADB:
            return DriverManager.getConnection(oraCS, oraUser, oraPwd);
        case MYSQLDB :
            return DriverManager.getConnection(mysqlCS, mysqlUser, mysqlPwd);
        default:
            return null;
        }
    }
}
```

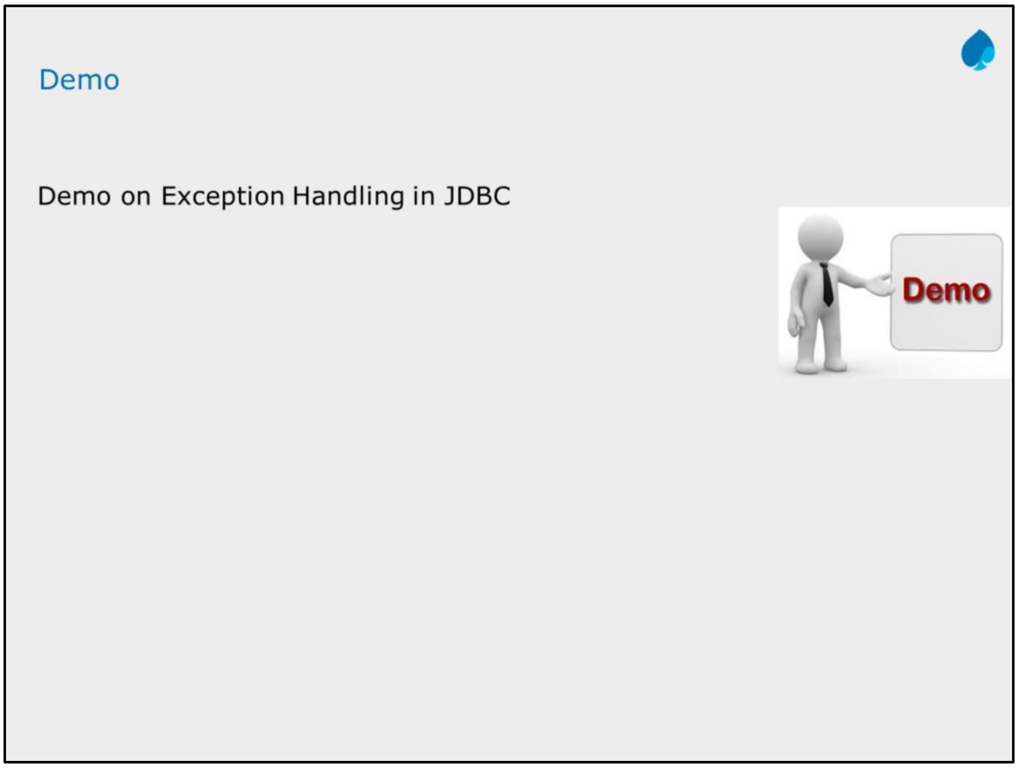


Handling JDBC Exceptions

JDBC Methods throw SQLException

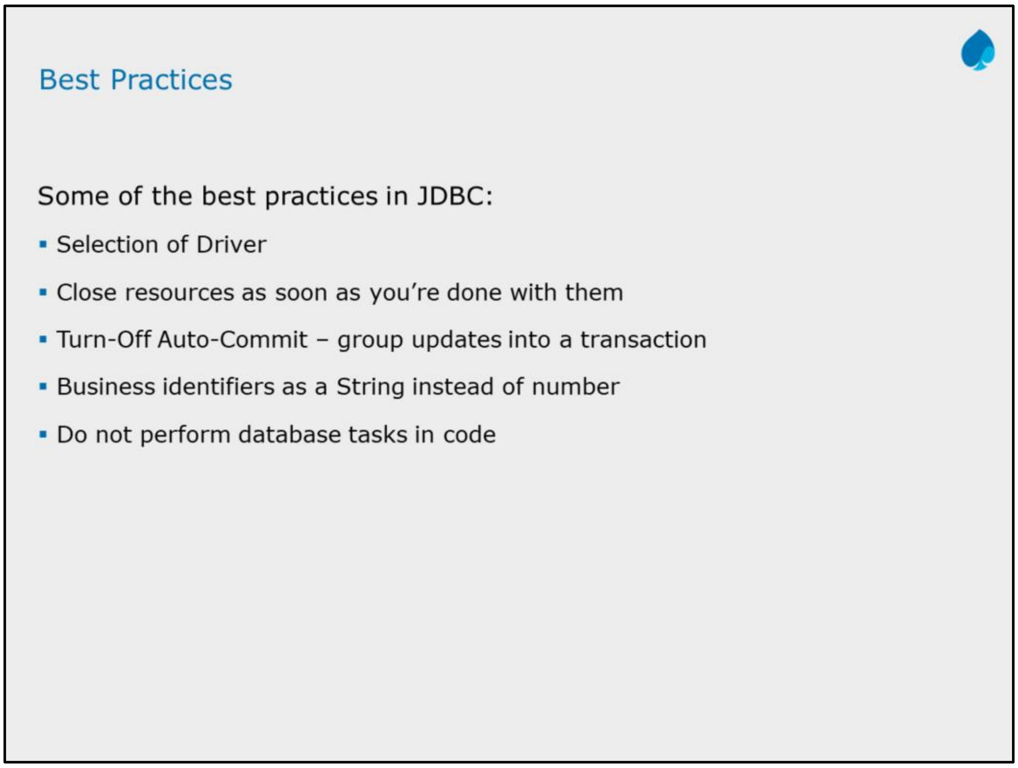
SQLException extended from Exception class

- getMessage()
- getLocalizedMessage()
- getErrorCode() : return int value which tells you what went wrong
- getSQLState() : return the state value for exception



Demo

Demo on Exception Handling in JDBC



Best Practices

Some of the best practices in JDBC:

- Selection of Driver
- Close resources as soon as you're done with them
- Turn-Off Auto-Commit – group updates into a transaction
- Business identifiers as a String instead of number
- Do not perform database tasks in code

JDBC Best Practices:
Following are some of the best practices used in JDBC:

Selection of Driver
Select a certified, high performance type 2 (Thin) JDBC driver and use the latest drivers release.
Close resources as soon as you are done with them (in finally)
For example: Statements, Connections, Resultsets, and so on.
Failure to do so will eventually cause the application to “hang”, and fail to respond to user actions.
Turn-Off Auto-Commit
It is best practice to execute the group of the statement together which are the part of the same transaction. It helps to avoid the overhead of data inconsistency.

JDBC Best Practices:

4. Business identifiers as a String instead of number

Many problem domains use numbers as business identifiers. Credit card numbers, bank account numbers, and the like, are often simply that - numbers. *It should always be kept in mind, however, that such items are primarily identifiers, and their numeric character is usually completely secondary*. Their primary function is to identify items in the problem domain, not to represent quantities of any sort.

For example: It almost never makes sense to operate on numeric business identifiers as true numbers - adding two account numbers, or multiplying an account number by -1, are meaningless operations. That is, one can strongly argue that modeling an account number as a Integer is inappropriate, simply because it does not behave as an Integer.

Furthermore, new business rules can occasionally force numeric business identifiers to be abandoned in favor of alphanumeric ones. It seems prudent to treat the content of such a business identifier as a business rule, subject to change. As usual, a program should minimize the ripple effects of such changes.

In addition, Strings can contain leading zeros, while numeric fields will remove them.

5. Do not perform database tasks in code

Databases are a mature technology, and they should be used to do as much work as possible. Do not do the following in code, if it can be done in SQL instead:

ordering (ORDER BY)
filtering based on criteria (WHERE)
joining tables (WHERE, JOIN)
summarizing (GROUP BY, COUNT, AVG, STDDEV)

Any corresponding task implemented entirely in code would very likely:

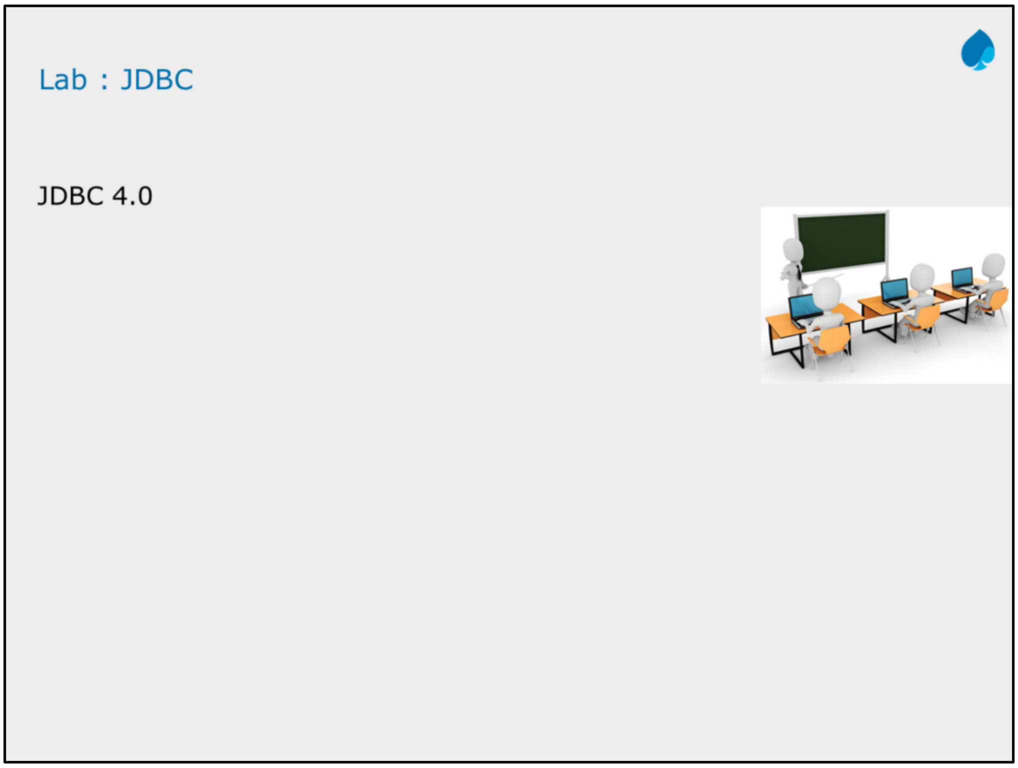
be *much* less robust and efficient
take longer to implement
require more maintenance effort

6. Use JDBC's PreparedStatement instead of Statement when possible for the following reasons:

It is in general more secure. When a Statement is constructed dynamically from user input, it is vulnerable to SQL injection attacks. PreparedStatement is not vulnerable in this way.

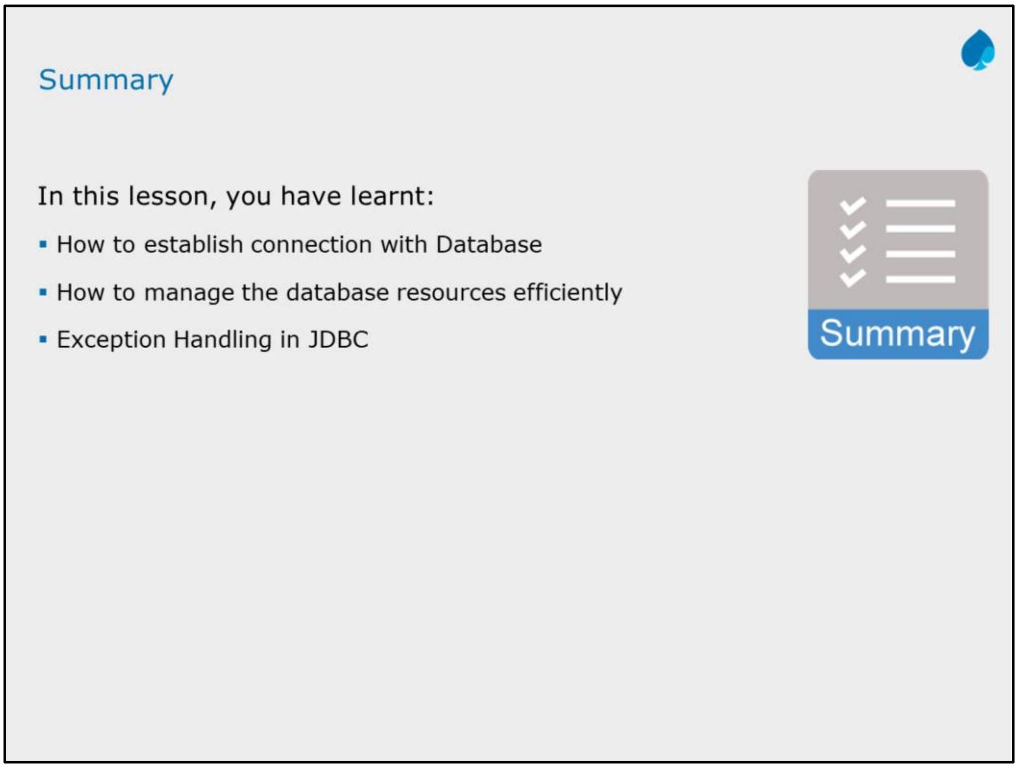
There is usually no need to worry about escaping special characters if repeated compilation is avoided, its performance is usually better.

In general, it seems safest to use a Statement only when the SQL is of fixed, known form, with no parameters



Lab : JDBC

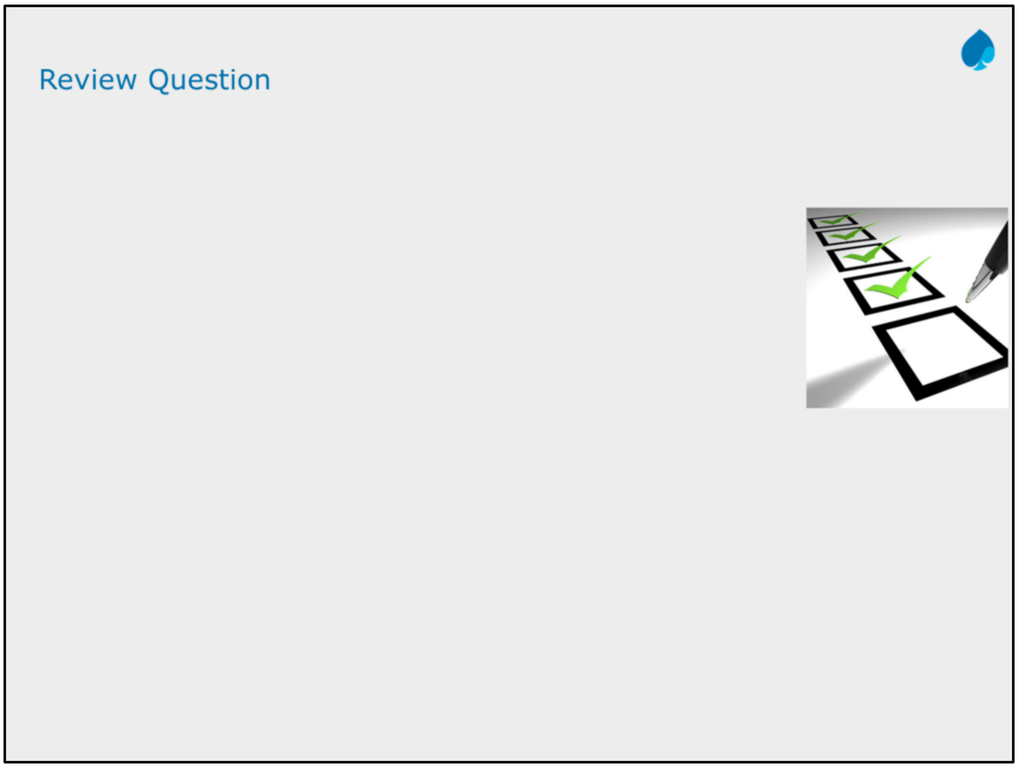
JDBC 4.0



Summary

In this lesson, you have learnt:

- How to establish connection with Database
- How to manage the database resources efficiently
- Exception Handling in JDBC



Review Question