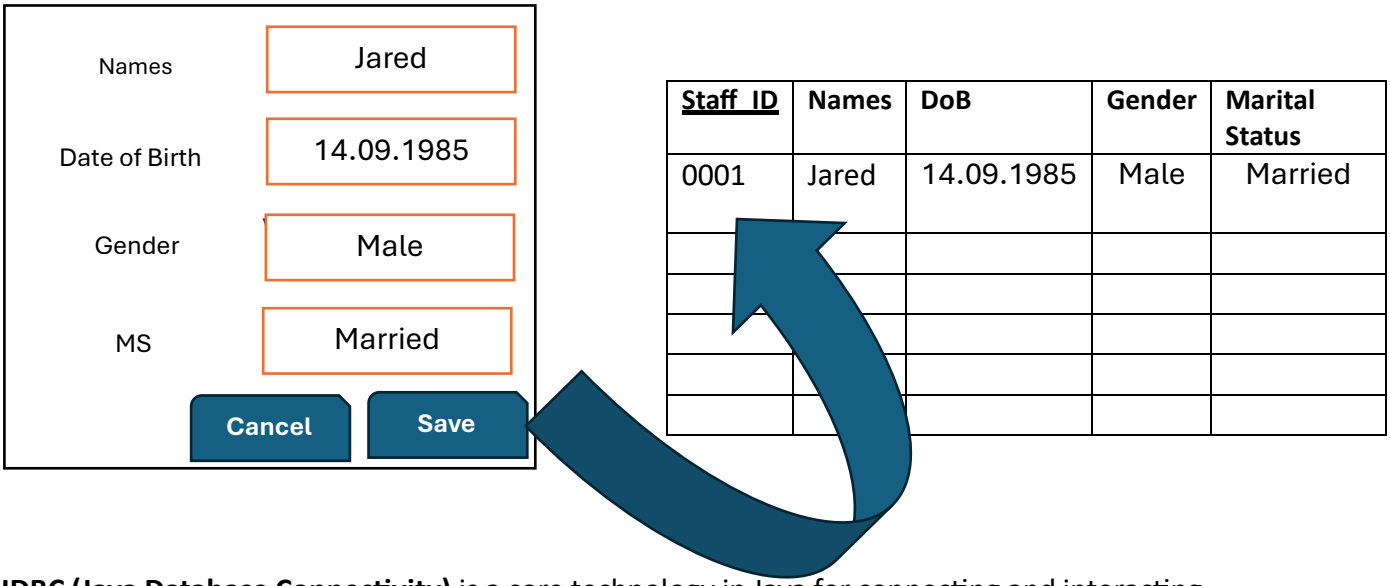
Week 5: JDBC & Database Integration

Introduction to JDBC



JDBC (Java Database Connectivity) is a core technology in Java for connecting and interacting with databases. It acts as a bridge between a Java application and a database, enabling Java programs to execute SQL queries, retrieve results, and modify data in a structured way. Think of JDBC as a translator that helps Java applications "speak" the language of databases, ensuring seamless interaction between your code and the data storage system.

Why Is JDBC Important?

- **Real-World Applications**: Most applications today require data storage and retrieval. For example, banking apps need to store transaction records, e-commerce platforms need to handle product and customer data, and social media apps need to manage user interactions. JDBC provides the mechanism for these operations.
- **Database Versatility**: JDBC supports various database management systems (DBMS) like **MySQL**, **PostgreSQL**, **Oracle**, and others. This makes it a flexible solution for developers who need to work with different types of databases.

Key Concepts of JDBC

1. JDBC API

The **JDBC API** is a collection of classes and interfaces that Java provides to enable database operations. Here's what you need to know:

- **Classes and Interfaces**:
    - **DriverManager**: Manages a list of database drivers. It is used to establish a connection between the Java application and the database.
    - **Connection**: Represents a connection to a specific database. Once a connection is established, SQL commands can be executed.

- **Statement**: Used to send basic SQL queries to the database.
- **PreparedStatement**: An extension of Statement that helps prevent SQL injection by safely handling query parameters.
- **ResultSet**: Holds the data retrieved from a database after executing a SELECT query. It acts as a cursor pointing to the current row of data.
- **CallableStatement**: Used to call stored procedures in the database.

Example: Think of the JDBC API as the tools you need to access a library. Connection is your library card, Statement is your request form for a book, and ResultSet is the book you receive that contains the information you need.

2. JDBC Drivers

A **JDBC Driver** is a software component that Java uses to interact with a database. It translates the API calls in the Java program into the database-specific calls that can be understood by the database management system.

Types of JDBC Drivers:

- **Type 1: JDBC-ODBC Bridge Driver**: Uses the ODBC driver to connect to the database. It is rarely used now due to performance issues.
- **Type 2: Native-API Driver**: Converts JDBC calls into database-specific API calls. It requires native database client libraries.
- **Type 3: Network Protocol Driver**: Communicates with the database over the network, translating JDBC calls into a database-independent protocol.
- **Type 4: Thin Driver (Pure Java Driver)**: The most commonly used driver type, written entirely in Java. It communicates directly with the database using the database's network protocol, without requiring native code. For example, **MySQL Connector/J** for MySQL databases.

Example: If JDBC is like a bridge connecting Java to databases, the **JDBC driver** is the specific type of bridge you need, depending on what kind of database you're crossing into (e.g., MySQL, Oracle, PostgreSQL).

How JDBC Works in a Java Program

Here's how the process flows when a Java program uses JDBC to interact with a database:

1. **Load the JDBC Driver**
    - The driver needs to be loaded so that Java knows which database it will connect to.
    - For most modern applications, this step is handled automatically when using **DriverManager**.

2. **Establish a Connection**
    - The program uses DriverManager.getConnection() to establish a connection to the database. This step requires the database's URL, username, and password.
3. **Create a Statement Object**
    - The program creates a Statement or PreparedStatement object, which will be used to send SQL queries to the database.
4. **Execute SQL Queries**
    - The statement object is used to execute SQL commands (e.g., SELECT, INSERT, UPDATE, DELETE).
    - If the query retrieves data, it returns a ResultSet object containing the results.
5. **Process the Results**
    - The ResultSet object is used to iterate through the rows of data retrieved from the database and process the information.
6. **Close the Connection**
    - The connection and other JDBC objects are closed to free up resources.

Full Example: JDBC in Action

Let's say you want to create a Java program that connects to a **MySQL** database and retrieves data from a table called books.

Code Example:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import java.sql.SQLException;

public class JDBCExample {
    public static void main(String[] args) {
        // JDBC URL, username, and password of MySQL server
        String url = "jdbc:mysql://localhost:3306/mydatabase"; // Replace 'mydatabase' with
your database name
        String username = "root"; // Replace with your database username
        String password = "password"; // Replace with your database password

        // Connection and statement initialization
        Connection connection = null;
        Statement statement = null;
```

```
        ResultSet resultSet = null;

        try {
            // Establish the connection
            connection = DriverManager.getConnection(url, username, password);
            System.out.println("Connected to the database!");

            // Create a statement to execute SQL queries
            statement = connection.createStatement();
            String query = "SELECT * FROM books"; // Replace 'books' with your table name

            // Execute the query and retrieve the result set
            resultSet = statement.executeQuery(query);

            // Iterate over the result set and print the data
            while (resultSet.next()) {
                System.out.println("Book ID: " + resultSet.getInt("id"));
                System.out.println("Title: " + resultSet.getString("title"));
                System.out.println("Author: " + resultSet.getString("author"));
                System.out.println("----------------------");
            }
        } catch (SQLException e) {
            System.out.println("Error: " + e.getMessage());
        } finally {
            // Close the result set, statement, and connection to free resources
            try {
                if (resultSet != null) resultSet.close();
                if (statement != null) statement.close();
                if (connection != null) connection.close();
            } catch (SQLException e) {
                System.out.println("Error closing resources: " + e.getMessage());
            }
        }
    }
}
```

```java
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import java.sql.SQLException;

public class JDBCExample {
    public static void main(String[] args) {
        // JDBC URL, username, and password of MySQL server
        String url = "jdbc:mysql://localhost:3306/mydatabase"; // Replace 'mydatabase' wi
        String username = "root"; // Replace with your database username
        String password = "password"; // Replace with your database password

        // Connection and statement initialization
        Connection connection = null;
        Statement statement = null;
        ResultSet resultSet = null;

        try {
            // Establish the connection
            connection = DriverManager.getConnection(url, username, password);
            System.out.println("Connected to the database!");

            // Create a statement to execute SQL queries
            statement = connection.createStatement();
            String query = "SELECT * FROM books"; // Replace 'books' with your table name

            // Execute the query and retrieve the result set
            resultSet = statement.executeQuery(query);

            // Iterate over the result set and print the data
            while (resultSet.next()) {
                System.out.println("Book ID: " + resultSet.getInt("id"));
                System.out.println("Title: " + resultSet.getString("title"));
                System.out.println("Author: " + resultSet.getString("author"));
                System.out.println("----------------------");
            }
        } catch (SQLException e) {
            System.out.println("Error: " + e.getMessage());
        } finally {
            // Close the result set, statement, and connection to free resources
            try {
                if (resultSet != null) resultSet.close();
                if (statement != null) statement.close();
                if (connection != null) connection.close();
            } catch (SQLException e) {
                System.out.println("Error closing resources: " + e.getMessage());
            }
        }
    }
}
```

Setting Up JDBC in NetBeans

To use **JDBC (Java Database Connectivity)** in your Java projects, you need to set up your development environment to connect your Java application to a database. Here's a simple guide on how to set up JDBC in **NetBeans IDE**.

Step 1: Add the JDBC Library

Before you can write code that connects to a database, you need to add the **JDBC driver** to your project. A JDBC driver is like a special tool that allows your Java application to talk to a specific type of database (e.g., MySQL, PostgreSQL, Oracle).

What Is a JDBC Driver?

Think of a JDBC driver as a bridge between your Java application and the database. Without this bridge, your program wouldn't know how to send and receive data from the database.

Example: MySQL Connector/J

For connecting to a **MySQL** database, you need the **MySQL Connector/J** driver. This is a Java library (a .jar file) that tells your Java program how to communicate with a MySQL database.

Steps to Add the JDBC Library in NetBeans

1. **Download the JDBC Driver (MySQL Connector/J)**
    - Visit the MySQL official website and download the **MySQL Connector/J** driver.
    - Choose the platform-independent version (typically a .zip or .tar.gz file).
    - Extract the downloaded file to get the mysql-connector-java-x.x.x.jar file (where x.x.x represents the version number).
2. **Add the JDBC Driver to Your NetBeans Project**
    - **Open NetBeans** and go to your Java project in which you want to set up JDBC.
    - **Right-click on the "Libraries" folder** under your project in the **Projects** tab.
    - Select **"Add JAR/Folder"**.
    - Browse to the location where you saved the mysql-connector-java-x.x.x.jar file, select it, and click **"Open"**.
    - The driver will now appear under the **Libraries** folder in your project, indicating that it has been added successfully.
3. **Verify the Addition**
    - Expand the **Libraries** section in your project to make sure the **MySQL Connector/J** library is listed.
    - This confirms that your project can now use the JDBC driver to connect to a MySQL database.

Why Is This Important?

Adding the JDBC driver to your project allows your Java program to use JDBC code to connect to and interact with a database. Without this step, your program would throw an

error when trying to establish a connection or execute SQL queries because it wouldn't know how to communicate with the database.

What Comes Next?

Now that you have added the JDBC driver to your project, you're ready to start writing Java code to establish a connection to your database, execute SQL commands, and retrieve data using JDBC.

Recap for Novice Students

- **JDBC Driver**: A library that helps your Java program talk to a database.
- **Adding the Driver**: You need to download the .jar file for the driver (e.g., **MySQL Connector/J** for MySQL) and add it to your project's libraries in NetBeans.
- **Result**: Your Java project can now connect to a database and perform operations like retrieving and updating data.

Example 1: Establishing a Database Connection

Full Code Example to Connect to a MySQL Database

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class DatabaseConnection {
    public static void main(String[] args) {
        // Database URL and credentials
        String url = "jdbc:mysql://localhost:3306/mydatabase"; // Replace 'mydatabase' with
your database name
        String username = "root"; // Replace with your database username
        String password = "password"; // Replace with your database password

        try {
            // Establish connection
            Connection connection = DriverManager.getConnection(url, username, password);
            System.out.println("Connected to the database successfully!");

            // Close the connection
            connection.close();
        } catch (SQLException e) {
            System.out.println("Connection failed: " + e.getMessage());
        }
    }
}
```

Example 2: Executing SQL Queries

Retrieving Data from the Database

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import java.sql.SQLException;

public class RetrieveData {
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/mydatabase";
        String username = "root";
        String password = "password";

        try {
            Connection connection = DriverManager.getConnection(url, username, password);
            Statement statement = connection.createStatement();
            String query = "SELECT * FROM books"; // Replace 'books' with your table name
            ResultSet resultSet = statement.executeQuery(query);

            while (resultSet.next()) {
                System.out.println("Book ID: " + resultSet.getInt("id"));
                System.out.println("Title: " + resultSet.getString("title"));
                System.out.println("Author: " + resultSet.getString("author"));
                System.out.println("----------------------");
            }
            resultSet.close();
            statement.close();
            connection.close();
        } catch (SQLException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

Exercises for Week 5

1. **Exercise 1: Create a Java Program to Insert Data**
    - Write a program that inserts a new record into your database table.
    - Use PreparedStatement to safely insert data and prevent SQL injection.
2. **Exercise 2: Update Existing Data**

- Create a Java program that updates an existing record in your database table (e.g., changing a book's title).

Week 6: Working with ResultSet & Stored Procedures

This week, we dive deeper into **retrieving and managing data from databases** using Java. When you execute SQL queries in a Java application, you need an efficient way to handle the data returned by the database. This is where **ResultSet** and **ResultSetMetaData** come into play. Additionally, we'll explore how to use **stored procedures** to execute precompiled SQL statements that can enhance the performance and security of your database interactions.

Introduction to ResultSet and ResultSetMetaData

When you execute a SELECT query in JDBC, the results are returned as a **ResultSet** object. This object acts as a table of data, where each row represents a record in the result, and each column represents a field in that record. It allows you to iterate over and access the retrieved data using methods provided by the ResultSet class.

Key Uses of ResultSet:

- Iterating through the rows of data returned by an SQL query.
- Accessing specific columns in each row.
- Navigating forward (and sometimes backward) through the result set, depending on the type of ResultSet.

Example: Imagine querying a database for a list of books. The ResultSet object holds this data, allowing you to process each book's information one by one.

ResultSetMetaData provides information about the structure of the ResultSet, such as:

- The number of columns.
- The names of the columns.
- The data types of each column.

Example: You might use ResultSetMetaData to dynamically determine the names and types of columns in a result set if you need to display the data in a generic format (e.g., a report or table).

Example Code: Using ResultSet and ResultSetMetaData in NetBeans

1. Retrieving Data with ResultSet:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import java.sql.SQLException;

public class ResultSetExample {
```

```
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/mydatabase"; // Replace with your DB name
        String username = "root"; // Replace with your username
        String password = "password"; // Replace with your password

        try (Connection connection = DriverManager.getConnection(url, username, password);
             Statement statement = connection.createStatement()) {

            // Execute a SELECT query
            String query = "SELECT id, title, author FROM books"; // Replace 'books' with your
table name
            ResultSet resultSet = statement.executeQuery(query);

            // Iterate through the ResultSet and print data
            while (resultSet.next()) {
                int id = resultSet.getInt("id");
                String title = resultSet.getString("title");
                String author = resultSet.getString("author");

                System.out.println("ID: " + id);
                System.out.println("Title: " + title);
                System.out.println("Author: " + author);
                System.out.println("----------------------");
            }

        } catch (SQLException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

2. Using ResultSetMetaData to Analyze Column Information:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.Statement;
import java.sql.SQLException;

public class ResultSetMetaDataExample {
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/mydatabase";
        String username = "root";
```

```
        String password = "password";

        try (Connection connection = DriverManager.getConnection(url, username, password);
             Statement statement = connection.createStatement()) {

            // Execute a SELECT query
            String query = "SELECT * FROM books";
            ResultSet resultSet = statement.executeQuery(query);
            ResultSetMetaData metaData = resultSet.getMetaData();

            // Print column information
            int columnCount = metaData.getColumnCount();
            System.out.println("Number of Columns: " + columnCount);

            for (int i = 1; i <= columnCount; i++) {
                System.out.println("Column " + i + ": " + metaData.getColumnName(i) +
                                   " (Type: " + metaData.getColumnTypeName(i) + ")");
            }

        } catch (SQLException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

Stored Procedures

Stored procedures are sets of precompiled SQL statements stored in the database. They can be executed by calling them from Java, allowing for better performance and easier maintenance. Using stored procedures can also improve security by preventing SQL injection attacks, as the SQL logic is stored on the server side.

Creating a Stored Procedure in MySQL:

```
DELIMITER //
CREATE PROCEDURE GetBooks()
BEGIN
    SELECT * FROM books;
END //
DELIMITER ;
```

Calling a Stored Procedure from Java:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.CallableStatement;
```

```
import java.sql.ResultSet;
import java.sql.SQLException;

public class StoredProcedureExample {
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/mydatabase";
        String username = "root";
        String password = "password";

        try (Connection connection = DriverManager.getConnection(url, username, password);
             CallableStatement callableStatement = connection.prepareCall("{CALL GetBooks()}"))
{

            // Execute the stored procedure
            ResultSet resultSet = callableStatement.executeQuery();

            // Print results
            while (resultSet.next()) {
                System.out.println("ID: " + resultSet.getInt("id"));
                System.out.println("Title: " + resultSet.getString("title"));
                System.out.println("Author: " + resultSet.getString("author"));
                System.out.println("----------------------");
            }

        } catch (SQLException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

Example 1: Using ResultSetMetaData

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.Statement;
import java.sql.SQLException;

public class ResultSetMetaDataExample {
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/mydatabase";
        String username = "root";
```

```
        String password = "password";

        try {
            Connection connection = DriverManager.getConnection(url, username, password);
            Statement statement = connection.createStatement();
            String query = "SELECT * FROM books";
            ResultSet resultSet = statement.executeQuery(query);
            ResultSetMetaData metaData = resultSet.getMetaData();

            int columnCount = metaData.getColumnCount();
            System.out.println("Column Count: " + columnCount);

            for (int i = 1; i <= columnCount; i++) {
                System.out.println("Column " + i + ": " + metaData.getColumnName(i) + " (Type: " +
metaData.getColumnTypeName(i) + ")");
            }
            resultSet.close();
            statement.close();
            connection.close();
        } catch (SQLException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

Assignment for Week 6: Advanced Database Interaction (30 Points)

This assignment focuses on developing your skills in using **JDBC**, **ResultSet**, **ResultSetMetaData**, and **stored procedures** in Java. You will practice creating and calling stored procedures and understanding the structure of data retrieved from a database. This assignment will be graded out of **30 points** and is designed to test both your understanding and practical application of these concepts.

Assignment Details

Exercise 1: Create and Use a Stored Procedure (15 Points)

Objective: Write a stored procedure that takes a book ID as an input and returns the details of the book. Implement a Java program that calls this stored procedure and displays the book's details.

Tasks:

1. **Create a Stored Procedure** (Database side):
    - Create a stored procedure in your database that accepts a book ID as a parameter and returns details like id, title, and author.

- Example structure of the stored procedure:

```
DELIMITER //
CREATE PROCEDURE GetBookDetails(IN bookID INT)
BEGIN
    SELECT id, title, author FROM books WHERE id = bookID;
END //
DELIMITER ;
```

2. **Implement a Java Program**:
    - Write a Java program in **NetBeans** that connects to your database and calls the stored procedure using CallableStatement.
    - Retrieve the book details from the stored procedure and display them in a formatted output.
    - Ensure you handle exceptions and close all resources properly.

Grading Criteria:

- Correctness of the stored procedure (5 points)
- Implementation of the Java program that calls the stored procedure (5 points)
- Proper display and formatting of the output (3 points)
- Error handling and resource management (2 points)

Example Java Code for Guidance:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.CallableStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

public class CallStoredProcedure {
   public static void main(String[] args) {
      String url = "jdbc:mysql://localhost:3306/mydatabase"; // Replace with your DB name
      String username = "root"; // Replace with your DB username
      String password = "password"; // Replace with your DB password

      try (Connection connection = DriverManager.getConnection(url, username, password);
         CallableStatement callableStatement = connection.prepareCall("{CALL
GetBookDetails(?)}")) {

         callableStatement.setInt(1, 1); // Replace with dynamic input as needed
```

```
            ResultSet resultSet = callableStatement.executeQuery();
            while (resultSet.next()) {
                System.out.println("ID: " + resultSet.getInt("id"));
                System.out.println("Title: " + resultSet.getString("title"));
                System.out.println("Author: " + resultSet.getString("author"));
                System.out.println("----------------------");
            }
        } catch (SQLException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

Exercise 2: Use ResultSetMetaData (15 Points)

Objective: Create a Java program that retrieves data from a database table and prints column information using **ResultSetMetaData**.

Tasks:

1. **Connect to the Database**:
    - Establish a connection to your database using DriverManager.
2. **Retrieve Data**:
    - Execute a SELECT query on a table (e.g., SELECT * FROM books) to get data.
3. **Use ResultSetMetaData**:
    - Retrieve and use ResultSetMetaData to print information about each column (e.g., column name, type, number of columns).
    - Display the retrieved column information along with some sample data from the ResultSet.

Grading Criteria:

- Connection and execution of the query (5 points)
- Proper use of ResultSetMetaData to display column information (5 points)
- Accurate output and formatting of data and column details (3 points)
- Exception handling and resource management (2 points)

Example Java Code for Guidance:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
```

```
import java.sql.ResultSetMetaData;
import java.sql.Statement;
import java.sql.SQLException;

public class ResultSetMetaDataExample {
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/mydatabase";
        String username = "root";
        String password = "password";

        try (Connection connection = DriverManager.getConnection(url, username, password);
             Statement statement = connection.createStatement();
             ResultSet resultSet = statement.executeQuery("SELECT * FROM books")) {

            ResultSetMetaData metaData = resultSet.getMetaData();
            int columnCount = metaData.getColumnCount();

            System.out.println("Column Information:");
            for (int i = 1; i <= columnCount; i++) {
                System.out.println("Column " + i + ": " + metaData.getColumnName(i) +
                                   " (Type: " + metaData.getColumnTypeName(i) + ")");
            }

            System.out.println("\nSample Data:");
            while (resultSet.next()) {
                for (int i = 1; i <= columnCount; i++) {
                    System.out.print(resultSet.getString(i) + " ");
                }
                System.out.println();
            }
        } catch (SQLException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```