# JDBC (Java Database Connectivity)

## What is JDBC?

- JDBC is an API (in `java.sql` package) that allows Java programs to connect to databases (like MySQL, Oracle, PostgreSQL, etc.), execute SQL queries, and process results.
- Acts as a **bridge** between Java application and database.

## Steps to Connect Database using JDBC

1. **Load Driver Class**

```
Class.forName("com.mysql.cj.jdbc.Driver");
```

2. **Establish Connection**

```
Connection con = DriverManager.getConnection(
    "jdbc:mysql://localhost:3306/mydb", "root", "password");
```

3. **Create Statement**

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

4. **Execute Query**

```
ResultSet rs = stmt.executeQuery("SELECT * FROM students");
while(rs.next()) {
    System.out.println(rs.getInt(1) + " " + rs.getString(2));
}
```

5. **Close Connection**

```
con.close();
```

### Example: Insert and Retrieve Data

```
import java.sql.*;

public class JDBCDemo {
    public static void main(String[] args) {
        try {
            // 1. Load Driver
            Class.forName("com.mysql.cj.jdbc.Driver");

            // 2. Connect
            Connection con = DriverManager.getConnection(
                "jdbc:mysql://localhost:3306/testdb", "root",
"password");

            // 3. Insert
            PreparedStatement ps = con.prepareStatement(
                "INSERT INTO student(id, name) VALUES(?, ?)");
            ps.setInt(1, 101);
            ps.setString(2, "Vikram");
            ps.executeUpdate();

            // 4. Select
            Statement stmt = con.createStatement();
            ResultSet rs = stmt.executeQuery("SELECT * FROM student");
            while (rs.next()) {
                System.out.println(rs.getInt("id") + " " +
rs.getString("name"));
            }

            // 5. Close
            con.close();
        } catch (Exception e) {
            e.printStackTrace();
```

```
        }
    }
}
```

### JDBC Lab Practices

1. Create a table `employee` and insert records using Java.
2. Retrieve all records from `employee`.
3. Update an employee's salary.
4. Delete a record by ID.
5. Create a login system using `username` and `password` fields.

# Threads in Java

## What is a Thread?

- A **thread** is a lightweight unit of a process (independent execution path).
- Allows **multitasking** (running multiple parts of a program simultaneously).
- Java provides thread support in `java.lang.Thread` and `java.lang.Runnable`.

## Ways to Create a Thread

### *(a) By Extending Thread Class*

```
class MyThread extends Thread {
    public void run() {
        System.out.println("Thread running: " +
Thread.currentThread().getName());
    }
}
```

```
public class ThreadDemo1 {
    public static void main(String[] args) {
        MyThread t1 = new MyThread();
        t1.start(); // start the thread
    }
}
```

### *(b) By Implementing Runnable Interface*

```
class MyTask implements Runnable {
    public void run() {
        System.out.println("Runnable running: " +
Thread.currentThread().getName());
    }
}

public class ThreadDemo2 {
    public static void main(String[] args) {
        Thread t1 = new Thread(new MyTask());
        t1.start();
    }
}
```

## Important Thread Methods

- `start()` → starts a new thread.
- `run()` → contains code to execute.
- `sleep(ms)` → pause thread for milliseconds.
- `join()` → wait for a thread to finish.
- `setPriority(int)` → change thread priority (1–10).
- `isAlive()` → check if thread is running.

## Example with Multiple Threads

```
class Task extends Thread {
    public void run() {
        for (int i = 1; i <= 5; i++) {
            System.out.println(Thread.currentThread().getName() + " -
" + i);
            try { Thread.sleep(500); } catch (Exception e) {}
        }
    }
}

public class MultiThreadDemo {
    public static void main(String[] args) {
        Task t1 = new Task();
        Task t2 = new Task();

        t1.start();
        t2.start();
    }
}
```

## Threads Lab Practices

1. Create a thread to print numbers from 1–10.
2. Create two threads: one prints even numbers, another prints odd numbers.
3. Use `sleep()` to make a clock thread print time every second.
4. Demonstrate `join()` by making one thread wait for another to finish.
5. Create a banking simulation (deposit/withdraw) using threads.