`NamedParameterJdbcTemplate`

Difference between **`JdbcTemplate`** and **`NamedParameterJdbcTemplate`** in Spring:

---

1. `JdbcTemplate`

- Uses **positional placeholders (`?`)** for SQL parameters.
- Parameters must be set in the exact order.
- Example:
- `String sql = "INSERT INTO employee (id, name, salary) VALUES (?, ?, ?)";`
- `jdbcTemplate.update(sql, 101, "Ravi", 50000.0);`
- Pros:
  - Simple, lightweight, widely used.
- Cons:
  - Readability suffers when many parameters are used.
  - Harder to maintain queries with many `?` placeholders.

---

2. `NamedParameterJdbcTemplate`

- Uses **named parameters (`:paramName`)** instead of `?`.
- Improves readability and maintainability of SQL queries.
- Example:
- `String sql = "INSERT INTO employee (id, name, salary) VALUES (:id, :name, :salary)";`
- `MapSqlParameterSource params = new MapSqlParameterSource();`
- `params.addValue("id", 102);`
- `params.addValue("name", "Sita");`
- `params.addValue("salary", 60000.0);`
-
- `namedParameterJdbcTemplate.update(sql, params);`
- Pros:
  - Clearer SQL statements with parameter names.
  - Order of parameters does not matter.
  - Useful when queries have many parameters or optional conditions.
- Cons:
  - Slightly more verbose to set up compared to `JdbcTemplate`.

Summary Table

Feature	`JdbcTemplate`	`NamedParameterJdbcTemplate`
Parameter Style	Positional (`?`)	Named (`:paramName`)
Readability with many params	Low	High
Order of parameters	Must match exactly	Independent of order
Use case	Simple queries with few params	Complex queries with many parameters

---

In practice:

- Use **`JdbcTemplate`** when queries are simple.
- Use **`NamedParameterJdbcTemplate`** when queries are complex, long, or dynamically built.

CRUD (Create, Read, Update, Delete) using **`NamedParameterJdbcTemplate`**

`Employee` and `Department` entities.

Step 1: Entities

```
// Department.java
public class Department {
    private int id;
    private String name;

    // Getters and Setters
    public int getId() { return id; }
    public void setId(int id) { this.id = id; }
    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
}
// Employee.java
public class Employee {
    private int id;
    private String name;
    private double salary;
    private int deptId;

    // Getters and Setters
    public int getId() { return id; }
    public void setId(int id) { this.id = id; }
    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
    public double getSalary() { return salary; }
    public void setSalary(double salary) { this.salary = salary;
}
    public int getDeptId() { return deptId; }
    public void setDeptId(int deptId) { this.deptId = deptId; }
}
```

□ Step 2: DAO Interface

```
// DepartmentDAO.java
public interface DepartmentDAO {
    void insert(Department dept);
    Department findById(int id);
    List<Department> findAll();
    void update(Department dept);
```

```
    void delete(int id);
}
// EmployeeDAO.java
public interface EmployeeDAO {
    void insert(Employee emp);
    Employee findById(int id);
    List<Employee> findAll();
    void update(Employee emp);
    void delete(int id);
}
```

Step 3: DAO Implementation with `NamedParameterJdbcTemplate`

```
// DepartmentDAOImpl.java
import
org.springframework.jdbc.core.namedparam.NamedParameterJdbcTempl
ate;
import
org.springframework.jdbc.core.namedparam.MapSqlParameterSource;
import org.springframework.jdbc.core.RowMapper;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.List;

public class DepartmentDAOImpl implements DepartmentDAO {

    private NamedParameterJdbcTemplate
namedParameterJdbcTemplate;

    public DepartmentDAOImpl(NamedParameterJdbcTemplate
namedParameterJdbcTemplate) {
        this.namedParameterJdbcTemplate =
namedParameterJdbcTemplate;
    }

    @Override
    public void insert(Department dept) {
        String sql = "INSERT INTO department (id, name) VALUES
(:id, :name)";
        MapSqlParameterSource params = new
MapSqlParameterSource()
                .addValue("id", dept.getId())
                .addValue("name", dept.getName());
        namedParameterJdbcTemplate.update(sql, params);
    }
```

```
    @Override
    public Department findById(int id) {
        String sql = "SELECT * FROM department WHERE id = :id";
        MapSqlParameterSource params = new
MapSqlParameterSource().addValue("id", id);
        return namedParameterJdbcTemplate.queryForObject(sql,
params, new DepartmentRowMapper());
    }

    @Override
    public List<Department> findAll() {
        String sql = "SELECT * FROM department";
        return namedParameterJdbcTemplate.query(sql, new
DepartmentRowMapper());
    }

    @Override
    public void update(Department dept) {
        String sql = "UPDATE department SET name = :name WHERE
id = :id";
        MapSqlParameterSource params = new
MapSqlParameterSource()
                .addValue("id", dept.getId())
                .addValue("name", dept.getName());
        namedParameterJdbcTemplate.update(sql, params);
    }

    @Override
    public void delete(int id) {
        String sql = "DELETE FROM department WHERE id = :id";
        MapSqlParameterSource params = new
MapSqlParameterSource().addValue("id", id);
        namedParameterJdbcTemplate.update(sql, params);
    }

    private static final class DepartmentRowMapper implements
RowMapper<Department> {
        @Override
        public Department mapRow(ResultSet rs, int rowNum)
throws SQLException {
            Department d = new Department();
            d.setId(rs.getInt("id"));
            d.setName(rs.getString("name"));
            return d;
        }
    }
}
```

```
// EmployeeDAOImpl.java
import
org.springframework.jdbc.core.namedparam.NamedParameterJdbcTempl
ate;
import
org.springframework.jdbc.core.namedparam.MapSqlParameterSource;
import org.springframework.jdbc.core.RowMapper;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.List;

public class EmployeeDAOImpl implements EmployeeDAO {

    private NamedParameterJdbcTemplate
namedParameterJdbcTemplate;

    public EmployeeDAOImpl(NamedParameterJdbcTemplate
namedParameterJdbcTemplate) {
        this.namedParameterJdbcTemplate =
namedParameterJdbcTemplate;
    }

    @Override
    public void insert(Employee emp) {
        String sql = "INSERT INTO employee (id, name, salary,
dept_id) " +
                     "VALUES (:id, :name, :salary, :deptId)";
        MapSqlParameterSource params = new
MapSqlParameterSource()
                .addValue("id", emp.getId())
                .addValue("name", emp.getName())
                .addValue("salary", emp.getSalary())
                .addValue("deptId", emp.getDeptId());
        namedParameterJdbcTemplate.update(sql, params);
    }

    @Override
    public Employee findById(int id) {
        String sql = "SELECT * FROM employee WHERE id = :id";
        MapSqlParameterSource params = new
MapSqlParameterSource().addValue("id", id);
        return namedParameterJdbcTemplate.queryForObject(sql,
params, new EmployeeRowMapper());
    }

    @Override
```

```
    public List<Employee> findAll() {
        String sql = "SELECT * FROM employee";
        return namedParameterJdbcTemplate.query(sql, new
EmployeeRowMapper());
    }

    @Override
    public void update(Employee emp) {
        String sql = "UPDATE employee SET name = :name, salary =
:salary, dept_id = :deptId " +
                     "WHERE id = :id";
        MapSqlParameterSource params = new
MapSqlParameterSource()
                .addValue("id", emp.getId())
                .addValue("name", emp.getName())
                .addValue("salary", emp.getSalary())
                .addValue("deptId", emp.getDeptId());
        namedParameterJdbcTemplate.update(sql, params);
    }

    @Override
    public void delete(int id) {
        String sql = "DELETE FROM employee WHERE id = :id";
        MapSqlParameterSource params = new
MapSqlParameterSource().addValue("id", id);
        namedParameterJdbcTemplate.update(sql, params);
    }

    private static final class EmployeeRowMapper implements
RowMapper<Employee> {
        @Override
        public Employee mapRow(ResultSet rs, int rowNum) throws
SQLException {
            Employee e = new Employee();
            e.setId(rs.getInt("id"));
            e.setName(rs.getString("name"));
            e.setSalary(rs.getDouble("salary"));
            e.setDeptId(rs.getInt("dept_id"));
            return e;
        }
    }
}
```

Step 4: Spring Configuration (`AppConfig.java`)

```
import javax.sql.DataSource;
```

```
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import
org.springframework.jdbc.core.namedparam.NamedParameterJdbcTempl
ate;
import
org.springframework.jdbc.datasource.DriverManagerDataSource;

@Configuration
public class AppConfig {

    @Bean
    public DataSource dataSource() {
        DriverManagerDataSource ds = new
DriverManagerDataSource();
        ds.setDriverClassName("com.mysql.cj.jdbc.Driver");
        ds.setUrl("jdbc:mysql://localhost:3306/testdb");
        ds.setUsername("root");
        ds.setPassword("root");
        return ds;
    }

    @Bean
    public NamedParameterJdbcTemplate
namedParameterJdbcTemplate(DataSource dataSource) {
        return new NamedParameterJdbcTemplate(dataSource);
    }

    @Bean
    public DepartmentDAO
departmentDAO(NamedParameterJdbcTemplate template) {
        return new DepartmentDAOImpl(template);
    }

    @Bean
    public EmployeeDAO employeeDAO(NamedParameterJdbcTemplate
template) {
        return new EmployeeDAOImpl(template);
    }
}
```

Step 5: Main Application

```
import
org.springframework.context.annotation.AnnotationConfigApplicati
onContext;
```

```
public class App {
    public static void main(String[] args) {
        AnnotationConfigApplicationContext ctx = new
AnnotationConfigApplicationContext(AppConfig.class);

        DepartmentDAO deptDAO =
ctx.getBean(DepartmentDAO.class);
        EmployeeDAO empDAO = ctx.getBean(EmployeeDAO.class);

        // Insert
        Department dept = new Department();
        dept.setId(1);
        dept.setName("IT");
        deptDAO.insert(dept);

        Employee emp = new Employee();
        emp.setId(101);
        emp.setName("Ravi");
        emp.setSalary(50000);
        emp.setDeptId(1);
        empDAO.insert(emp);

        // Read
        System.out.println("Employee: " +
empDAO.findById(101).getName());

        // Update
        emp.setName("Ravi Kumar");
        empDAO.update(emp);

        // Delete
        empDAO.delete(101);
        deptDAO.delete(1);

        ctx.close();
    }
}
```

This implementation covers **full CRUD with `NamedParameterJdbcTemplate`** for both entities.

SQL is more **readable & maintainable** than using `JdbcTemplate`.