Spring-JDBC

This Java class is a **Spring Configuration class** that sets up **Spring JDBC** with **transaction management** and **dependency injection**. It is used to connect a **Spring application** to a **MySQL database**.

Key Annotations Used

1 @Configuration

- Marks this class as a **Spring configuration class**.
- It acts as a replacement for XML-based configuration.

2 @EnableTransactionManagement

- Enables **Spring's transaction management**.
- Allows methods annotated with @Transactional to use **automatic transaction handling**.

3 @ComponentScan(basePackages = "com.coforge.springjdbc")

- Tells Spring to **scan the package** "com.coforge.springjdbc" for components like @Component, @Service, and @Repository classes.

Bean Definitions in SpringJdbcConfig

This class defines **three beans** that are required for Spring JDBC to work.

1 DataSource Bean (Database Connection)

```
@Bean
    public DataSource dataSource() {
        DriverManagerDataSource dataSource = new DriverManagerDataSource();

        // MySQL JDBC configuration
        dataSource.setDriverClassName("com.mysql.cj.jdbc.Driver");
```

```
        dataSource.setUrl("jdbc:mysql://localhost:3306/mydb2");
        dataSource.setUsername("root");
        dataSource.setPassword("root");

        return dataSource;
    }
```

✔ **Creates a DataSource bean** using DriverManagerDataSource.
✔ **Configures the database connection** to MySQL (mydb2 on localhost:3306).
✔ Uses **username = root**, **password = root**.

This is the primary connection to the database.

---

2 JdbcTemplate Bean (For Query Execution)

```
@Bean
    public JdbcTemplate jdbcTemplate(DataSource dataSource) {
        return new JdbcTemplate(dataSource);
    }
```

✔ **Creates a JdbcTemplate bean**.
✔ Injects the dataSource to connect to the database.
✔ Provides methods for executing SQL queries (insert, update, query, etc.).

This is used in DAO (Data Access Objects) for database operations.

3PlatformTransactionManager Bean (Transaction Management)

```
@Bean
    public PlatformTransactionManager
transactionManager(DataSource dataSource) {
        return new DataSourceTransactionManager(dataSource);
    }
```

✔ **Creates a PlatformTransactionManager bean** to manage database transactions.
✔ Enables **@Transactional** support to ensure data consistency.

This ensures that database operations follow the ACID principles (Atomicity, Consistency, Isolation, Durability).

Summary of What This Class Does

Configures **database connection** (DataSource).
Provides **JdbcTemplate** for executing SQL queries.
Enables **transaction management** (@Transactional).
Uses **annotation-based configuration** instead of XML.

Example Usage in a DAO Class

```
@Repository
public class UserDao {

    private final JdbcTemplate jdbcTemplate;

    public UserDao(JdbcTemplate jdbcTemplate) {
        this.jdbcTemplate = jdbcTemplate;
    }

    @Transactional
    public void createUser(String username, String email) {
        String insertUserSQL = "INSERT INTO users (username, email) VALUES (?, ?)";
        jdbcTemplate.update(insertUserSQL, username, email);


        if (username.equals("error")) {
            throw new RuntimeException("Simulated Error: Rollback transaction");
        }
    }
}
```

Spring will automatically inject JdbcTemplate, and database operations will work smoothly.