Java Full Stack Lab Manual - JDBC Experiment

Experiment: Using Spring JdbcTemplate to Manage Database and Transactions

Aim

To demonstrate how to use Spring JdbcTemplate for performing database operations and managing transactions using DataSourceTransactionManager.

Technologies Used

Java, Spring Framework (Core + JDBC + TX), Maven, MySQL, Apache Tomcat (optional), Eclipse or IntelliJ

Directory Structure

```
SpringJdbcTemplateDemo/
|- src/main/java/
|  |- com/example/config/
|     |- AppConfig.java
|  |- com/example/dao/
|     |- StudentDAO.java
|  |- com/example/model/
|     |- Student.java
|  |- com/example/main/
|     |- AppMain.java
|- pom.xml
```

pom.xml (Dependencies)

```
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>5.3.20</version>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-jdbc</artifactId>
        <version>5.3.20</version>
    </dependency>
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>8.0.33</version>
    </dependency>
</dependencies>
```

Database Table

Java Full Stack Lab Manual - JDBC Experiment

```
CREATE DATABASE college;
USE college;

CREATE TABLE student (
    id INT PRIMARY KEY,
    name VARCHAR(50),
    address VARCHAR(100)
);
```

AppConfig.java

```
@Configuration
@ComponentScan(basePackages = "com.example")
public class AppConfig {
    @Bean
    public DataSource dataSource() {
        DriverManagerDataSource ds = new DriverManagerDataSource();
        ds.setDriverClassName("com.mysql.cj.jdbc.Driver");
        ds.setUrl("jdbc:mysql://localhost:3306/college");
        ds.setUsername("root");
        ds.setPassword("root");
        return ds;
    }

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

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

Java Full Stack Lab Manual - JDBC Experiment

Student.java

```
public class Student {
    private int id;
    private String name;
    private String address;

    // Getters and Setters
}
```

StudentDAO.java

```
@Repository
public class StudentDAO {

    @Autowired
    private JdbcTemplate jdbcTemplate;

    @Transactional
    public void insertAndUpdate(Student s1, Student s2) {
           jdbcTemplate.update("INSERT INTO student VALUES (?, ?, ?)", s1.getId(), s1.getName(),
s1.getAddress());
           jdbcTemplate.update("INSERT INTO student VALUES (?, ?, ?)", s2.getId(), s2.getName(),
s2.getAddress());

        if (true) {
            throw new RuntimeException("Simulated failure");
        }

             jdbcTemplate.update("UPDATE student SET address = ? WHERE id = ?", "Bangalore",
s1.getId());
    }
}
```

Java Full Stack Lab Manual - JDBC Experiment

AppMain.java

```
public class AppMain {
    public static void main(String[] args) {
                                    AnnotationConfigApplicationContext     context     =     new
AnnotationConfigApplicationContext(AppConfig.class);
        StudentDAO dao = context.getBean(StudentDAO.class);

        Student s1 = new Student(); s1.setId(1); s1.setName("Madhu"); s1.setAddress("Hyderabad");
        Student s2 = new Student(); s2.setId(2); s2.setName("Sravani"); s2.setAddress("Guntur");

        try {
            dao.insertAndUpdate(s1, s2);
        } catch (Exception e) {
            System.out.println("Transaction failed and rolled back: " + e.getMessage());
        }

        context.close();
    }
}
```

Expected Output

```
Transaction failed and rolled back: Simulated failure
```

Result

Successfully implemented Spring JdbcTemplate with transaction management.