* [Home](https://www.javainuse.com/home)
* [Java](https://www.javainuse.com/)
* [Spring](https://www.javainuse.com/)
* [Angular](https://www.javainuse.com/)
* [Full Stack](https://www.javainuse.com/)
* [Apache Camel](https://www.javainuse.com/)
* [Cloud Frameworks](https://www.javainuse.com/)
* [Messaging](https://www.javainuse.com/)
* [Drools](https://www.javainuse.com/)
* [Search Engine](https://www.javainuse.com/)
* [Hazelcast](https://www.javainuse.com/hazelcast)
* [DevOps](https://www.javainuse.com/)

Search Tutorials

Top of Form

|  |  |  |  |
| --- | --- | --- | --- |
| |  |  | | --- | --- | |  |  | |  |

Bottom of Form

Spring Boot Transactions - Understanding Transaction Propagation

In previous tutorial - [Spring Boot Transaction Management Example](https://www.javainuse.com/spring/boot-transaction) we saw what are transactions and implemented declarative transaction management. In this tutorial we will be understanding what is propagation and its different types. In next tutorial we will be looking at [Transaction Rollbacks for checked exceptions using Spring Boot.](https://www.javainuse.com/spring/boot-rollback)

Spring Boot Transaction Management - Table of Contents

[Spring Boot Transaction Management Example](https://www.javainuse.com/spring/boot-transaction)

[**Spring Boot Transactions - Understanding Transaction Propagation**](https://www.javainuse.com/spring/boot-transaction-propagation)

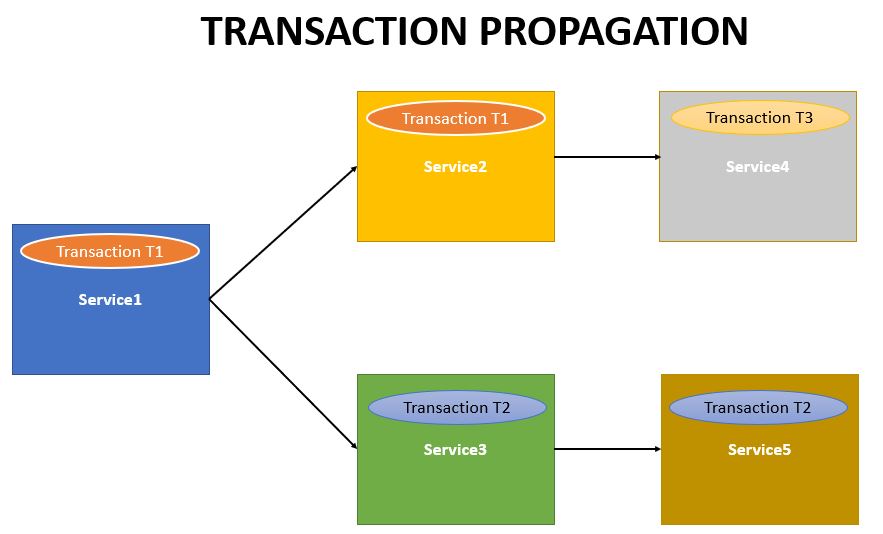
[Spring Boot Transactions - Understanding Transaction Rollbacks](https://www.javainuse.com/spring/boot-rollback)

[Spring Boot Transactions - Understanding Transaction Isolation](https://www.javainuse.com/spring/boot-transaction-isolation)

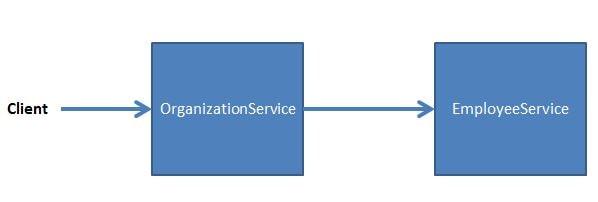
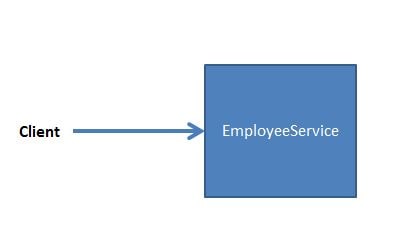
Video

This tutorial is explained in the below Youtube Video.

Lets Begin-

What is Transaction Propagation?  
Any application involves a number of services or components making a call to other services or components. Transaction Propagation indicates if any component or service will or will not participate in transaction and how will it **behave if the calling calling component/service already has or does not have a transaction created already.**  


We will be making use of the Spring Boot Transaction Project we developed in previous chapter. It had the Organization service which makes a call to the Employee Service and the Health Insurance Service.  
Also [previous example we had added transaction annotation only to the Organization service.](https://www.javainuse.com/spring/boot-transaction)  
But suppose the user wants to call the Employee Service in both ways i.e.

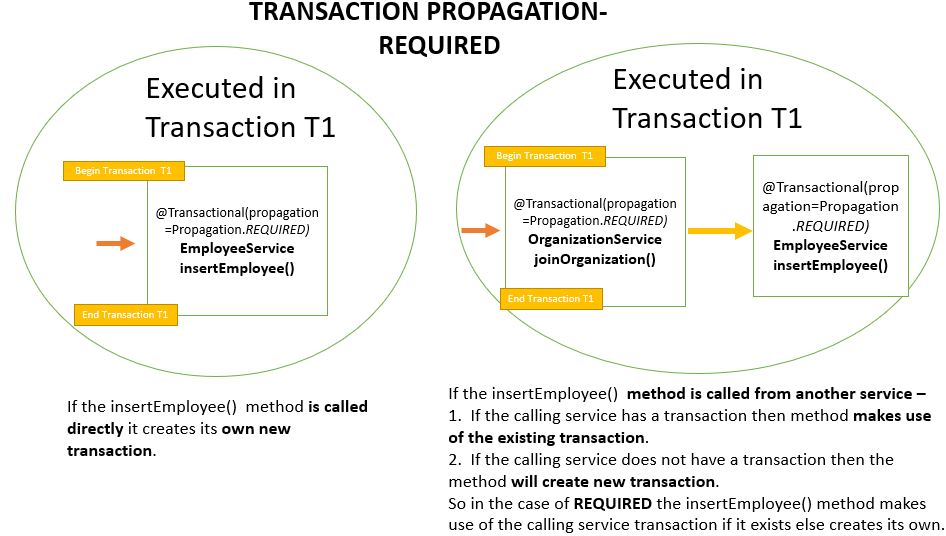
* Call using Organization service  
  
* Call the the Employee Service directly.  
  

As the Employee Service may also be called directly we will need to use Transaction annotation with Employee Service also. So both the services - Organization Service and the Employee Service will be using Transaction annotation.

We will be looking at the various propagation scenarios by observing the behaviour of the Organization and Employee service. There are six types of Transaction Propagations-

* **REQUIRED**
* **SUPPORTS**
* **NOT\_SUPPORTED**
* **REQUIRES\_NEW**
* **NEVER**
* **MANDATORY**

Transaction Propagation - **REQUIRED** (Default Transaction Propagation)

  
Here both the Organization Service and the Employee Service have the transaction propagation defined as **Required**. This is the default Transaction Propagation.  
**Code-**  
The Organization Service will be as follows-

package com.javainuse.service.impl;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

import com.javainuse.model.Employee;

import com.javainuse.model.EmployeeHealthInsurance;

import com.javainuse.service.EmployeeService;

import com.javainuse.service.HealthInsuranceService;

import com.javainuse.service.OrganizationService;

@Service

**@Transactional**

public class OrganzationServiceImpl implements OrganizationService {

@Autowired

EmployeeService employeeService;

@Autowired

HealthInsuranceService healthInsuranceService;

@Override

public void joinOrganization(Employee employee, EmployeeHealthInsurance employeeHealthInsurance) {

employeeService.insertEmployee(employee);

if (employee.getEmpId().equals("emp1")) {

throw new RuntimeException("thowing exception to test transaction rollback");

}

healthInsuranceService.registerEmployeeHealthInsurance(employeeHealthInsurance);

}

@Override

public void leaveOrganization(Employee employee, EmployeeHealthInsurance employeeHealthInsurance) {

employeeService.deleteEmployeeById(employee.getEmpId());

healthInsuranceService.deleteEmployeeHealthInsuranceById(employeeHealthInsurance.getEmpId());

}

}

The Employee Service will be as follows-

package com.javainuse.service.impl;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

import com.javainuse.dao.EmployeeDao;

import com.javainuse.model.Employee;

import com.javainuse.service.EmployeeService;

@Service

**@Transactional**

public class EmployeeServiceImpl implements EmployeeService {

@Autowired

EmployeeDao employeeDao;

@Override

public void insertEmployee(Employee employee) {

employeeDao.insertEmployee(employee);

}

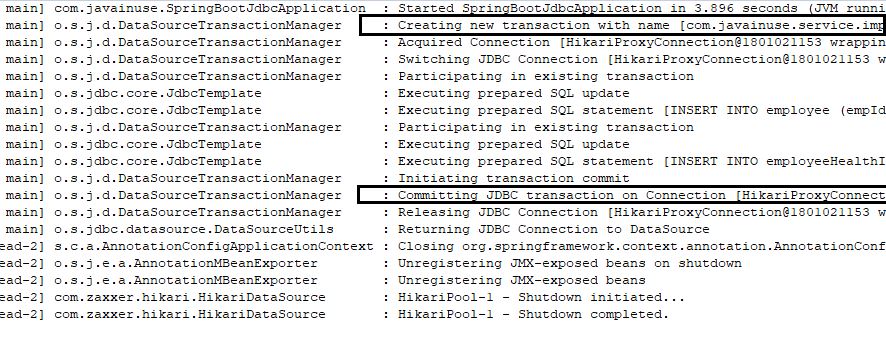
@Override

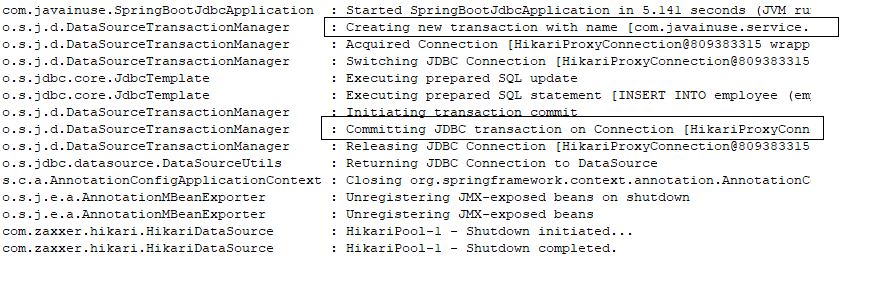
public void deleteEmployeeById(String empid) {

employeeDao.deleteEmployeeById(empid);

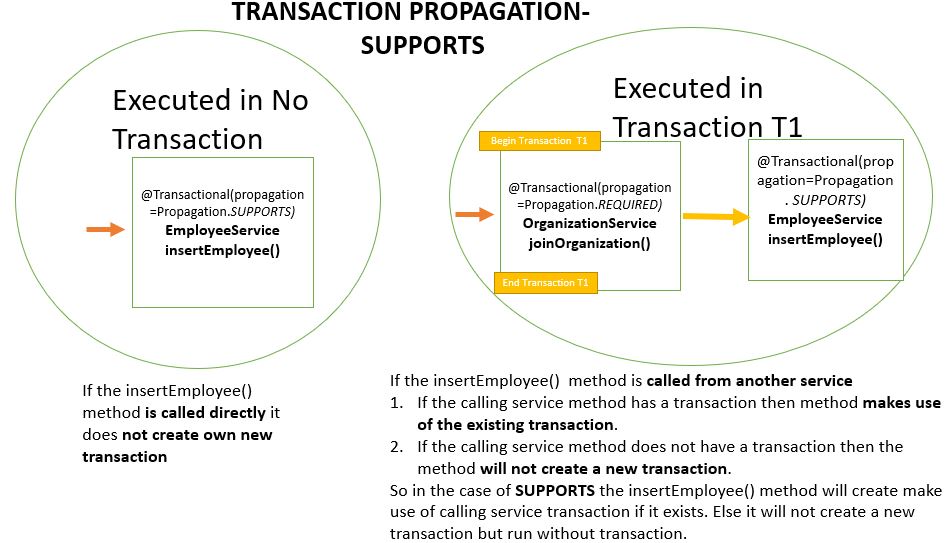
}

}

**Output**  
**EmployeeService called using OrganizationService -**  


**EmployeeService called directly -**  


Transaction Propagation - **SUPPORTS**

  
Here both the Organization Service has the transaction propagation defined as **Required** while Employee Service the transaction propagation is defined as **Supports**.  
**Code-**  
The Organization Service will be as follows-

package com.javainuse.service.impl;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

import com.javainuse.model.Employee;

import com.javainuse.model.EmployeeHealthInsurance;

import com.javainuse.service.EmployeeService;

import com.javainuse.service.HealthInsuranceService;

import com.javainuse.service.OrganizationService;

@Service

**@Transactional**

public class OrganzationServiceImpl implements OrganizationService {

@Autowired

EmployeeService employeeService;

@Autowired

HealthInsuranceService healthInsuranceService;

@Override

public void joinOrganization(Employee employee, EmployeeHealthInsurance employeeHealthInsurance) {

employeeService.insertEmployee(employee);

if (employee.getEmpId().equals("emp1")) {

throw new RuntimeException("thowing exception to test transaction rollback");

}

healthInsuranceService.registerEmployeeHealthInsurance(employeeHealthInsurance);

}

@Override

public void leaveOrganization(Employee employee, EmployeeHealthInsurance employeeHealthInsurance) {

employeeService.deleteEmployeeById(employee.getEmpId());

healthInsuranceService.deleteEmployeeHealthInsuranceById(employeeHealthInsurance.getEmpId());

}

}

The Employee Service will be as follows-

package com.javainuse.service.impl;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

import com.javainuse.dao.EmployeeDao;

import com.javainuse.model.Employee;

import com.javainuse.service.EmployeeService;

@Service

**@Transactional(propagation=Propagation.SUPPORTS)**

public class EmployeeServiceImpl implements EmployeeService {

@Autowired

EmployeeDao employeeDao;

@Override

public void insertEmployee(Employee employee) {

employeeDao.insertEmployee(employee);

}

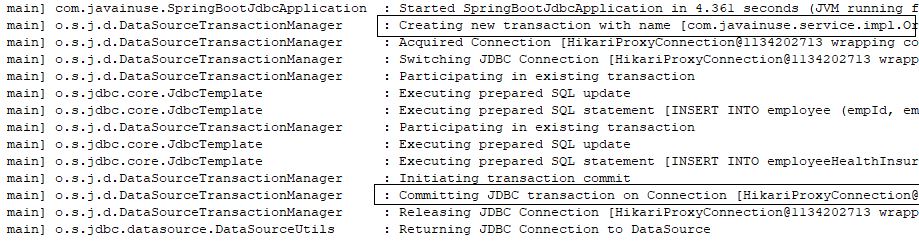
@Override

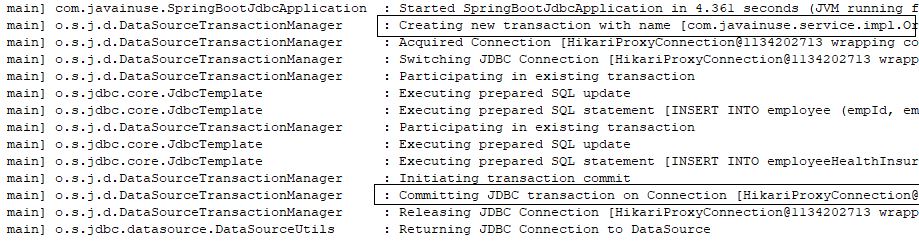
public void deleteEmployeeById(String empid) {

employeeDao.deleteEmployeeById(empid);

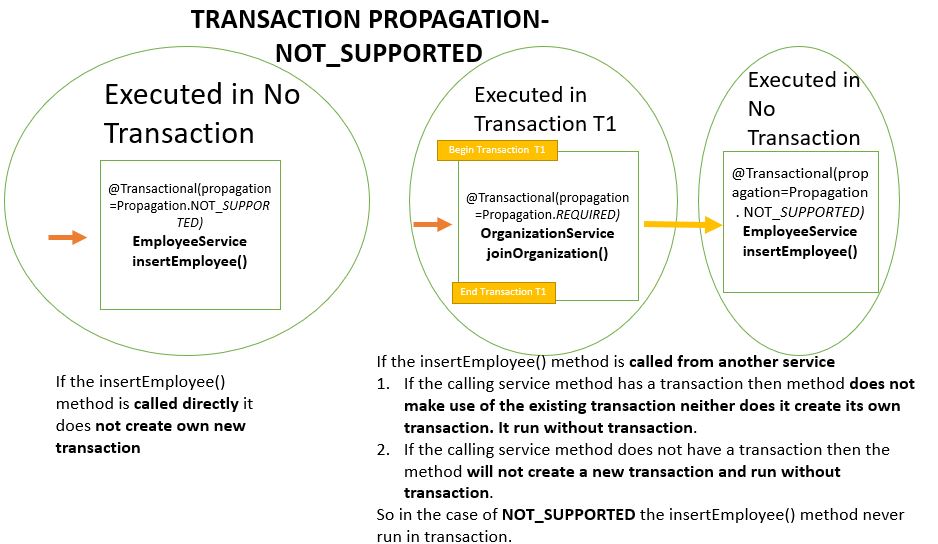
}

}

**Output**  
**EmployeeService called using OrganizationService -** 

**EmployeeService called directly -**

Transaction Propagation - **NOT\_SUPPORTED**

  
Here for the Organization Service we have defined the transaction propagation as **REQUIRED** and the Employee Service have the transaction propagation defined as **NOT\_SUPPORTED**  
**Code-**  
The Organization Service will be as follows-

package com.javainuse.service.impl;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

import com.javainuse.model.Employee;

import com.javainuse.model.EmployeeHealthInsurance;

import com.javainuse.service.EmployeeService;

import com.javainuse.service.HealthInsuranceService;

import com.javainuse.service.OrganizationService;

@Service

**@Transactional**

public class OrganzationServiceImpl implements OrganizationService {

@Autowired

EmployeeService employeeService;

@Autowired

HealthInsuranceService healthInsuranceService;

@Override

public void joinOrganization(Employee employee, EmployeeHealthInsurance employeeHealthInsurance) {

employeeService.insertEmployee(employee);

if (employee.getEmpId().equals("emp1")) {

throw new RuntimeException("thowing exception to test transaction rollback");

}

healthInsuranceService.registerEmployeeHealthInsurance(employeeHealthInsurance);

}

@Override

public void leaveOrganization(Employee employee, EmployeeHealthInsurance employeeHealthInsurance) {

employeeService.deleteEmployeeById(employee.getEmpId());

healthInsuranceService.deleteEmployeeHealthInsuranceById(employeeHealthInsurance.getEmpId());

}

}

The Employee Service will be as follows-

package com.javainuse.service.impl;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

import com.javainuse.dao.EmployeeDao;

import com.javainuse.model.Employee;

import com.javainuse.service.EmployeeService;

@Service

**@Transactional(propagation=Propagation.NOT\_SUPPORTED)**

public class EmployeeServiceImpl implements EmployeeService {

@Autowired

EmployeeDao employeeDao;

@Override

public void insertEmployee(Employee employee) {

employeeDao.insertEmployee(employee);

}

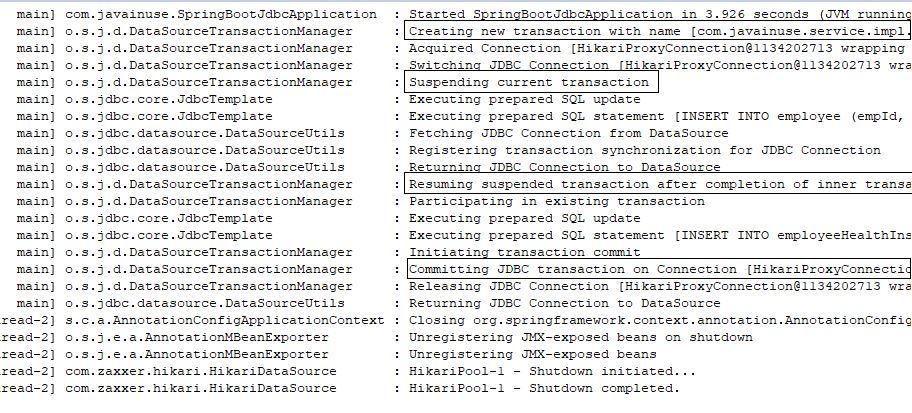
@Override

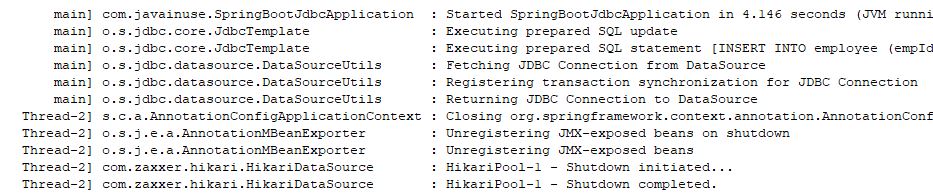
public void deleteEmployeeById(String empid) {

employeeDao.deleteEmployeeById(empid);

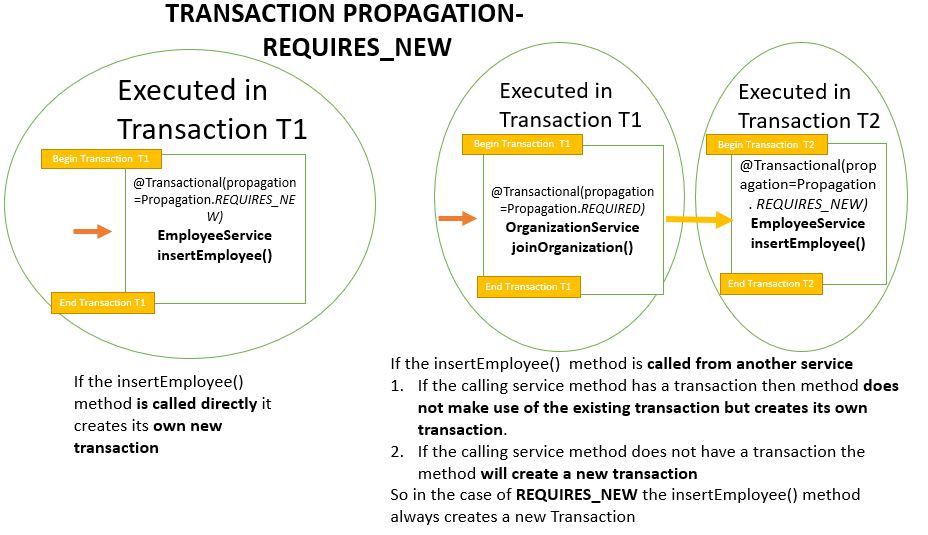
}

}

**Output**  
**EmployeeService called using OrganizationService -**  


**EmployeeService called directly -**  


Transaction Propagation - **REQUIRES\_NEW**

  
Here for the Organization Service we have defined the transaction propagation as **REQUIRED** and the Employee Service have the transaction propagation defined as **REQUIRES\_NEW**  
**Code-**  
The Organization Service will be as follows-

package com.javainuse.service.impl;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

import com.javainuse.model.Employee;

import com.javainuse.model.EmployeeHealthInsurance;

import com.javainuse.service.EmployeeService;

import com.javainuse.service.HealthInsuranceService;

import com.javainuse.service.OrganizationService;

@Service

**@Transactional**

public class OrganzationServiceImpl implements OrganizationService {

@Autowired

EmployeeService employeeService;

@Autowired

HealthInsuranceService healthInsuranceService;

@Override

public void joinOrganization(Employee employee, EmployeeHealthInsurance employeeHealthInsurance) {

employeeService.insertEmployee(employee);

if (employee.getEmpId().equals("emp1")) {

throw new RuntimeException("thowing exception to test transaction rollback");

}

healthInsuranceService.registerEmployeeHealthInsurance(employeeHealthInsurance);

}

@Override

public void leaveOrganization(Employee employee, EmployeeHealthInsurance employeeHealthInsurance) {

employeeService.deleteEmployeeById(employee.getEmpId());

healthInsuranceService.deleteEmployeeHealthInsuranceById(employeeHealthInsurance.getEmpId());

}

}

The Employee Service will be as follows-

package com.javainuse.service.impl;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

import com.javainuse.dao.EmployeeDao;

import com.javainuse.model.Employee;

import com.javainuse.service.EmployeeService;

@Service

**@Transactional(propagation=Propagation.REQUIRES\_NEW)**

public class EmployeeServiceImpl implements EmployeeService {

@Autowired

EmployeeDao employeeDao;

@Override

public void insertEmployee(Employee employee) {

employeeDao.insertEmployee(employee);

}

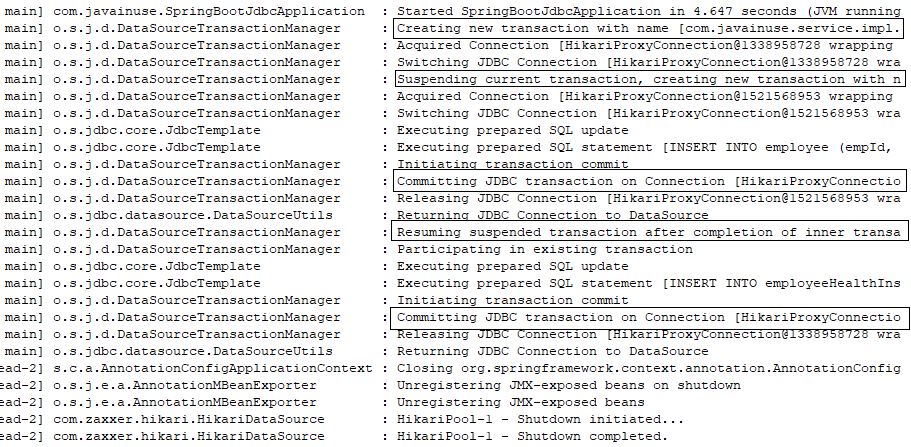
@Override

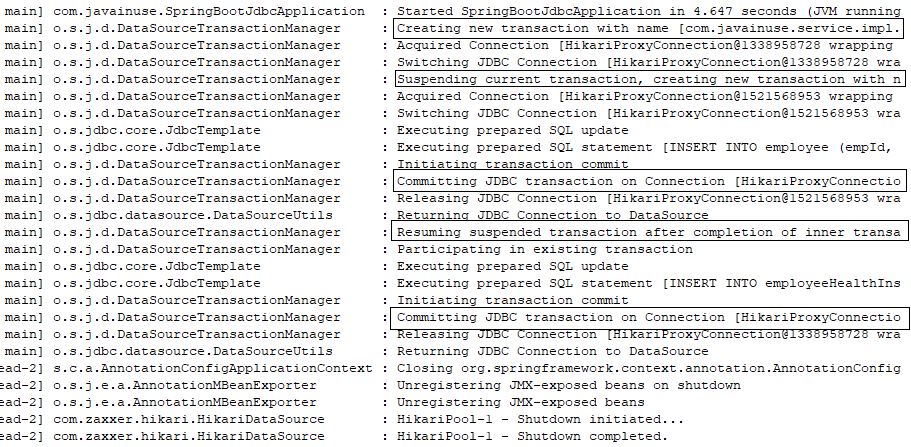
public void deleteEmployeeById(String empid) {

employeeDao.deleteEmployeeById(empid);

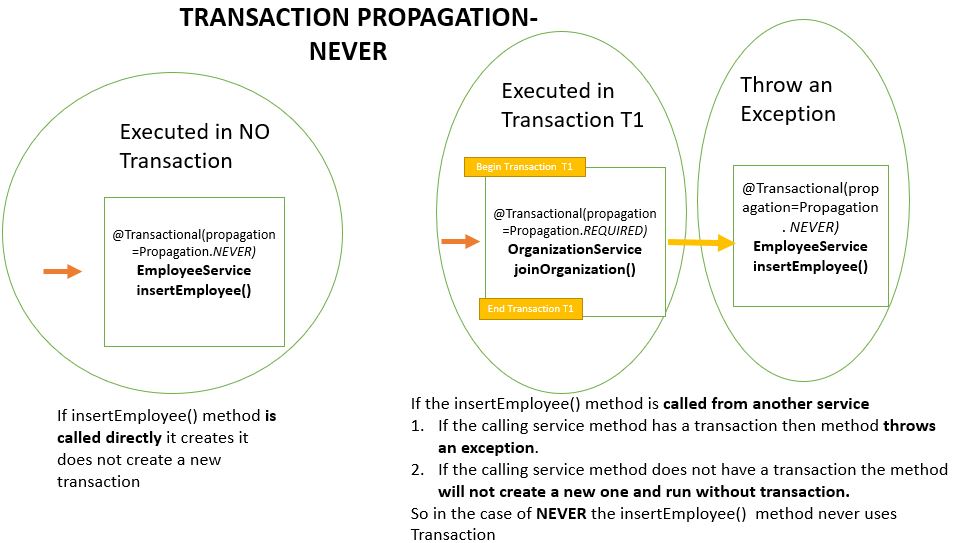
}

}

**Output**  
**EmployeeService called using OrganizationService -**  


**EmployeeService called directly -**  


Transaction Propagation - **NEVER**

  
Here for the Organization Service we have defined the transaction propagation as **REQUIRED** and the Employee Service have the transaction propagation defined as **NEVER**s  
**Code-**  
The Organization Service will be as follows-

package com.javainuse.service.impl;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

import com.javainuse.model.Employee;

import com.javainuse.model.EmployeeHealthInsurance;

import com.javainuse.service.EmployeeService;

import com.javainuse.service.HealthInsuranceService;

import com.javainuse.service.OrganizationService;

@Service

**@Transactional**

public class OrganzationServiceImpl implements OrganizationService {

@Autowired

EmployeeService employeeService;

@Autowired

HealthInsuranceService healthInsuranceService;

@Override

public void joinOrganization(Employee employee, EmployeeHealthInsurance employeeHealthInsurance) {

employeeService.insertEmployee(employee);

if (employee.getEmpId().equals("emp1")) {

throw new RuntimeException("thowing exception to test transaction rollback");

}

healthInsuranceService.registerEmployeeHealthInsurance(employeeHealthInsurance);

}

@Override

public void leaveOrganization(Employee employee, EmployeeHealthInsurance employeeHealthInsurance) {

employeeService.deleteEmployeeById(employee.getEmpId());

healthInsuranceService.deleteEmployeeHealthInsuranceById(employeeHealthInsurance.getEmpId());

}

}

The Employee Service will be as follows-

package com.javainuse.service.impl;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

import com.javainuse.dao.EmployeeDao;

import com.javainuse.model.Employee;

import com.javainuse.service.EmployeeService;

@Service

**@Transactional(propagation=Propagation.NEVER)**

public class EmployeeServiceImpl implements EmployeeService {

@Autowired

EmployeeDao employeeDao;

@Override

public void insertEmployee(Employee employee) {

employeeDao.insertEmployee(employee);

}

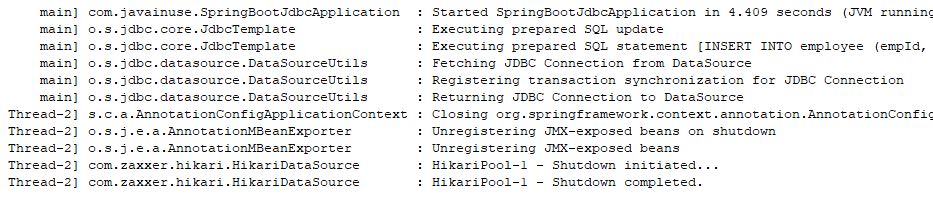
@Override

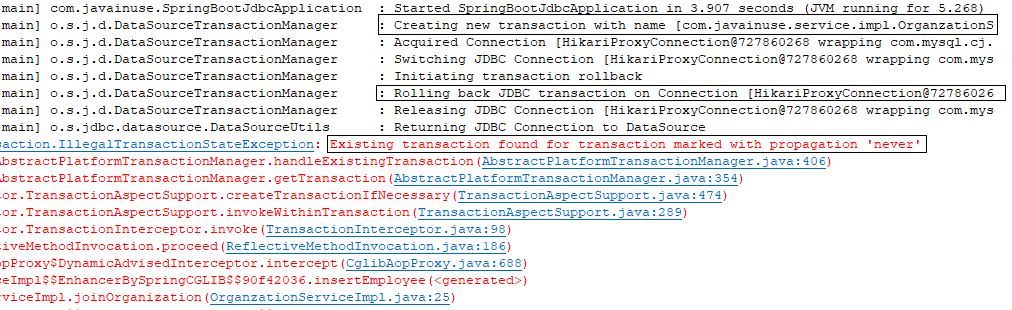
public void deleteEmployeeById(String empid) {

employeeDao.deleteEmployeeById(empid);

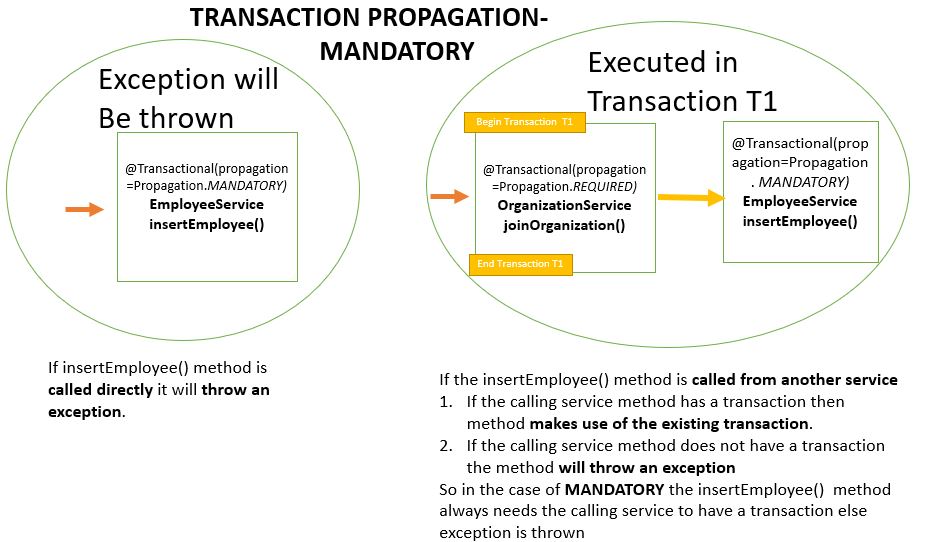
}

}

**Output**  
**EmployeeService called using OrganizationService -**  


**EmployeeService called directly -**  


Transaction Propagation - **MANDATORY**

  
Here for the Organization Service we have defined the transaction propagation as **REQUIRED** and the Employee Service have the transaction propagation defined as **MANDATORY**  
**Code-**  
The Organization Service will be as follows-

package com.javainuse.service.impl;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

import com.javainuse.model.Employee;

import com.javainuse.model.EmployeeHealthInsurance;

import com.javainuse.service.EmployeeService;

import com.javainuse.service.HealthInsuranceService;

import com.javainuse.service.OrganizationService;

@Service

**@Transactional**

public class OrganzationServiceImpl implements OrganizationService {

@Autowired

EmployeeService employeeService;

@Autowired

HealthInsuranceService healthInsuranceService;

@Override

public void joinOrganization(Employee employee, EmployeeHealthInsurance employeeHealthInsurance) {

employeeService.insertEmployee(employee);

if (employee.getEmpId().equals("emp1")) {

throw new RuntimeException("thowing exception to test transaction rollback");

}

healthInsuranceService.registerEmployeeHealthInsurance(employeeHealthInsurance);

}

@Override

public void leaveOrganization(Employee employee, EmployeeHealthInsurance employeeHealthInsurance) {

employeeService.deleteEmployeeById(employee.getEmpId());

healthInsuranceService.deleteEmployeeHealthInsuranceById(employeeHealthInsurance.getEmpId());

}

}

The Employee Service will be as follows-

package com.javainuse.service.impl;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

import com.javainuse.dao.EmployeeDao;

import com.javainuse.model.Employee;

import com.javainuse.service.EmployeeService;

@Service

**@Transactional(propagation=Propagation.MANDATORY)**

public class EmployeeServiceImpl implements EmployeeService {

@Autowired

EmployeeDao employeeDao;

@Override

public void insertEmployee(Employee employee) {

employeeDao.insertEmployee(employee);

}

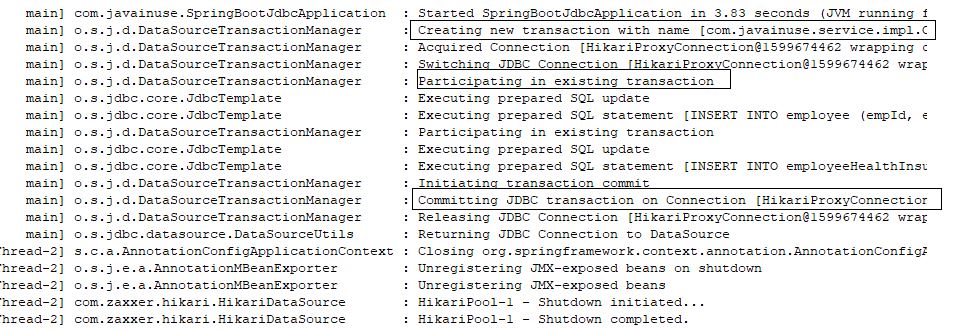
@Override

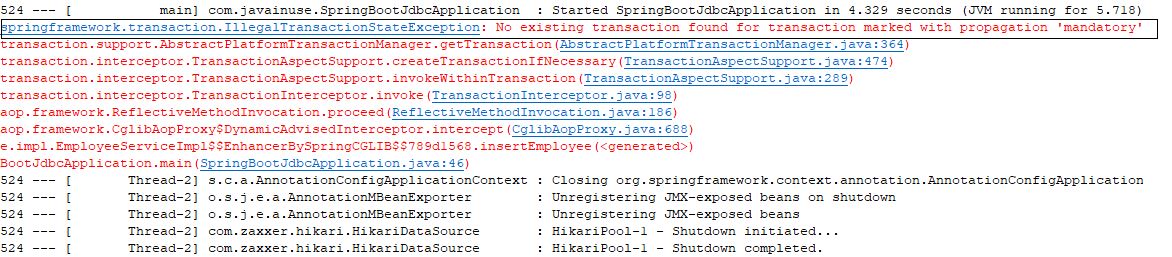
public void deleteEmployeeById(String empid) {

employeeDao.deleteEmployeeById(empid);

}

}

**Output**  
**EmployeeService called using OrganizationService -**  


**EmployeeService called directly -**  
  
So the summary will be as follows-

|  |  |
| --- | --- |
| **Propagation** | **Behaviour** |
| **REQUIRED** | **Always executes in a transaction.** If there is any existing transaction it uses it. If none exists then only a new one is created |
| **SUPPORTS** | **It may or may not run in a transaction.** If current transaction exists then it is supported. If none exists then gets executed with out transaction. |
| **NOT\_SUPPORTED** | **Always executes without a transaction.** If there is any existing transaction it gets suspended |
| **REQUIRES\_NEW** | **Always executes in a new transaction.** If there is any existing transaction it gets suspended |
| **NEVER** | **Always executes with out any transaction.** It throws an exception if there is an existing transaction |
| **MANDATORY** | **Always executes in a transaction.** If there is any existing transaction it is used. If there is no existing transaction it will throw an exception. |

Popular Posts

* [E-commerce Website - Online Book Store using Angular 8 + Spring Boot](https://www.javainuse.com/fullstack/ecommerce)
* [Spring Boot +JSON Web Token(JWT) Hello World Example](https://www.javainuse.com/spring/boot-jwt)
* [Angular 7 + Spring Boot Application Hello World Example](https://www.javainuse.com/spring/ang7-hello)
* [Build a Real Time Chat Application using Spring Boot + WebSocket + RabbitMQ](https://www.javainuse.com/spring/boot-websocket-chat)
* [Pivotal Cloud Foundry Tutorial - Deploy Spring Boot Application Hello World Example](https://www.javainuse.com/pcf/pcf-hello)
* [Deploying Spring Based WAR Application to Docker](https://www.javainuse.com/devOps/docker/docker-war)
* [EIP patterns using Apache Camel](https://www.javainuse.com/camel/camel_EIP)
* [Spring Cloud- Netflix Eureka + Ribbon Simple Example](https://www.javainuse.com/spring/spring_ribbon)
* [Spring Cloud- Netflix Hystrix Circuit Breaker Simple Example](https://www.javainuse.com/spring/spring_hystrix_circuitbreaker)
* [Spring Boot + Swagger Example Hello World Example](https://www.javainuse.com/spring/boot_swagger)
* [Spring Boot Batch Simple example](https://www.javainuse.com/spring/bootbatch)
* [Spring Boot + Apache Kafka Example](https://www.javainuse.com/spring/spring-boot-apache-kafka-hello-world)
* [Spring Boot Admin Simple Example](https://www.javainuse.com/spring/boot-admin)
* [Spring Boot Security - Introduction to OAuth](https://www.javainuse.com/spring/spring-boot-oauth-introduction)
* [Spring Boot OAuth2 Part 1 - Getting The Authorization Code](https://www.javainuse.com/spring/spring-boot-oauth-authorization-code)
* [Spring Boot OAuth2 Part 2 - Getting The Access Token And Using it to Fetch Data.](https://www.javainuse.com/spring/spring-boot-oauth-access-token)
* [JBoss Drools Hello World-Stateful Knowledge Session using KieSession](https://www.javainuse.com/drools_hello_kie)
* [Understand Drools Stateful vs Stateless Knowledge Session](https://www.javainuse.com/drools_states)
* [JBoss Drools- Understanding Drools Decision Table using Simple Example](https://www.javainuse.com/drools/drools_decision)

See Also

* [Spring Boot Interview Questions](https://www.javainuse.com/spring/SpringBootInterviewQuestions)
* [Spring Batch Interview Questions](https://www.javainuse.com/spring/sprbatch_interview)
* [Spring AOP Interview Questions](https://www.javainuse.com/spring/spring-AOP-interview-quesions)
* [Angular 2 Interview Questions](https://www.javainuse.com/angular/ang2_intvw)
* [Apache Camel Interview Questions](https://www.javainuse.com/camel/Apache_Camel_Questions)
* [JBoss Fuse Interview Questions](https://www.javainuse.com/camel/JBoss_Fuse_Questions)
* [Drools Interview Questions](https://www.javainuse.com/drools/drools_intvw)
* [Java 8 Interview Questions](https://www.javainuse.com/java/java8_intvw)
* [Spring Cloud Interview Questions](https://www.javainuse.com/spring/spring-cloud-interview-questions)
* [Microservices Interview Questions](https://www.javainuse.com/spring/microservices-interview-quesions)
* [Java HashMap and ConcurrentHashMap Interview Questions](https://www.javainuse.com/java/java_map_intvw)
* [Mule ESB frequently asked interview questions](https://www.javainuse.com/misc/muleintvw)
* [Apache Kafka Interview Questions](https://www.javainuse.com/misc/apache-kafka-interview-questions)
* [Tosca Testing Tool Interview Questions](https://www.javainuse.com/misc/tosca-testing-tool-interview-questions)
* [Top Maven Build Tool Interview Questions](https://www.javainuse.com/misc/maven-interview-questions)
* [Top Gradle Build Tool Interview Questions](https://www.javainuse.com/misc/gradle-interview-questions)
* [Miscellaneous Topics](https://www.javainuse.com/misc)

© Copyright JavaInUse. All Rights Reserved.  
[Privacy Policy](https://www.javainuse.com/privacy)