Spring Boot Transaction Propagation Example

In the previous tutorial, we've seen the Spring Boot Transaction Management Example using Declarative Transaction Management.
In this tutorial, we're going to understand what is *transaction propagation* and it's different types.
In the next tutorial, we'll look at Transaction Rollbacks for the checked exceptions using Spring Boot.

What is Transaction Propagation?

There are many components/services involved in the enterprise application for any given request to get the job done. Some of these components/services mark the transaction boundary (start/end) that will be used in the corresponding component and its sub-components. For this transaction boundary of components/services, Transaction Propagation specifies whether or not the corresponding component will participate in the transaction and what happens if the calling component/service has or does not already have a transaction created/started.

In enterprise application, there are many services or components involved to compeleted the job/request. Transaction Propogation specifies whether or not the current component/service will participate in the transaction, and what if the calling component/service has or does not already have a transaction created/started.

Transaction Propagation Types: There are six types of Transaction Propagation:
REQUIRED is Default Transaction Propagation.

- REQUIRED
- SUPPORTS
- NOT_SUPPORTED
- REQUIRES_NEW
- NEVER
- MANDATORY

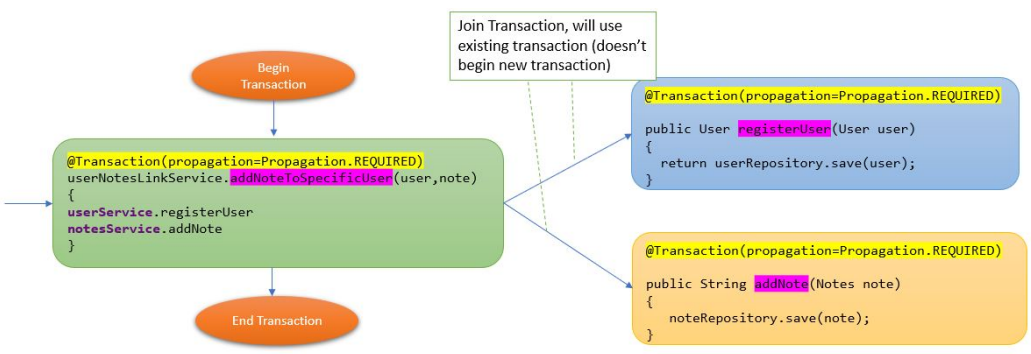
Propagation	**Behaviour**

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.

In this tutorial, we're going to look at the different scenarios to understand transaction propagation types in detail.

1. ## REQUIRED Propagation:

CASE 1:



We will take example code from Spring Boot Transaction Management, as seen registerUser and addNote service methods are getting called from addNoteToSpecficUser. So here we have added @transaction at the top of addNoteToSpecficUser method, without any propagation parameter, so by default Spring will consider REQUIRED type. In this case, if calling service addNoteToSpecficUser having REQUIRED propagation, so registerUser and addNote will make use of existing transaction created from addNoteToSpecficUser.

If we add @transaction(REQUIRED) at the top of registerUser and addNote method, it will consider calling service transaction propagation type, else it'll creates it's own transaction if calling service has no transaction created.

Code: From below main class, calling addNoteToSpecificUser method.

```
package com.techgeeknext;

import com.techgeeknext.modal.Notes;
import com.techgeeknext.modal.User;
import com.techgeeknext.service.UserNotesLinkService;
import com.techgeeknext.service.UserService;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootAppl
import org.springframework.context.ApplicationContext;
import org.springframework.data.jpa.repository.config.Enable
import org.springframework.web.bind.annotation.RestControlle

@SpringBootApplication
@EnableJpaAuditing
@RestController
public class TransactionManagementApplication {

    public static void main(String[] args) throws Exception
        ApplicationContext context = SpringApplication.run(T
        UserNotesLinkService userNotesLinkService = context.

        UserService userService = context.getBean(UserServic

        /* TEST DATA FOR TRANSACTION MANAGEMENT EXAMPLE*/
        // create new user
        User user = new User();
        user.setUserMail("techgeeknext@gmail.com");
        user.setUserPass("12345678jhg");

        //create new note
        Notes note = new Notes();
        note.setTitle("Test Note");
        note.setMessage("Test Message");

        //link above new user with above note
        userNotesLinkService.addNoteToSpecificUser(user, not
    }
}
```

UserNotesLinkServiceImpl: It has addNoteToSpecificUser implementation, it calls registerUser from UserService and addNote methods from NotesService.

```
package com.techgeeknext.service.impl;

import com.techgeeknext.modal.Notes;
import com.techgeeknext.modal.User;
import com.techgeeknext.service.NotesService;
import com.techgeeknext.service.UserNotesLinkService;
import com.techgeeknext.service.UserService;
import com.techgeeknext.util.ApplicationConstants;
import org.springframework.beans.factory.annotation.Autowire
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Propagatio
import org.springframework.transaction.annotation.Transactio
@Service
public class UserNotesLinkServiceImpl implements UserNotesLi
    @Autowired
    private UserService userService;

    @Autowired
    private NotesService notesService;

    @Override
    @Transactional(propagation= Propagation.REQUIRED)
    public String addNoteToSpecificUser(User user, Notes not
        //create new user
        User createdUser = userService.registerUser(user);
        Notes dbNote = new Notes();
        dbNote.setTitle(note.getTitle());
        dbNote.setMessage(note.getMessage());
        //set created user to note
        dbNote.setUserDetails(createdUser);
        //persist new note
        notesService.addNote(dbNote);
        return ApplicationConstants.ADDED_NOTE_DESC;
    }
}
```

UserServiceImpl: It contains implementation of registerUser method.

```
package com.techgeeknext.service.impl;

import com.techgeeknext.modal.User;
import com.techgeeknext.repository.UserRepository;
import com.techgeeknext.service.UserService;
import org.springframework.beans.factory.annotation.Autowire
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Propagatio
import org.springframework.transaction.annotation.Transactio

@Service
public class UserServiceImpl implements  UserService{
    @Autowired
    UserRepository userRepository;


    @Override
    @Transactional(propagation= Propagation.REQUIRED)
    public User registerUser(User user)
    {
        return userRepository.save(user);
    }
}
```

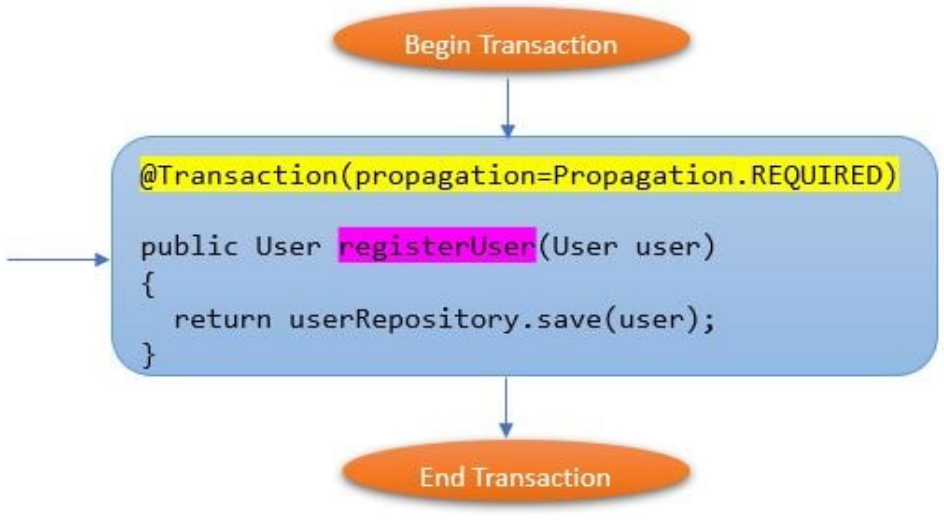
NotesServiceImpl: It contains implementation of addNote method.

```
package com.techgeeknext.service.impl;
import java.util.List;

import javax.validation.Valid;

import com.techgeeknext.modal.NoteBase;
import com.techgeeknext.modal.Notes;
import com.techgeeknext.modal.User;
import com.techgeeknext.repository.NotesRepository;
import com.techgeeknext.repository.UserRepository;
import com.techgeeknext.service.NotesService;
import com.techgeeknext.util.ApplicationConstants;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import org.springframework.beans.factory.annotation.Autowire
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactio
import org.springframework.transaction.annotation.Propagatio

@Service
public class NotesServiceImpl implements NotesService {

    @Autowired
    private NotesRepository noteRepository;

    @Override
    @Transactional(propagation= Propagation.REQUIRED)
    public String addNote(Notes note){
        noteRepository.save(note);
        return ApplicationConstants.ADDED_NOTE_DESC;
    }
}
```

Output

```
2020-07-29 12:23:24.733  INFO 25992 --- [  restartedMain] c.t.TransactionManagementApplication     : Started TransactionManagementApplication in 4.697
 seconds (JVM running for 5.156)
2020-07-29 12:23:24.741 DEBUG 25992 --- [  restartedMain] o.s.orm.jpa.JpaTransactionManager        : Creating new transaction with name [com.techgeeknext
 .service.impl.UserNotesLinkServiceImpl.addNoteToSpecificUser]: PROPAGATION_REQUIRED,ISOLATION_DEFAULT
2020-07-29 12:23:24.741 DEBUG 25992 --- [  restartedMain] o.s.orm.jpa.JpaTransactionManager        : Opened new EntityManager [SessionImpl(4619534<open>)]
 for JPA transaction
2020-07-29 12:23:24.747 DEBUG 25992 --- [  restartedMain] o.s.orm.jpa.JpaTransactionManager        : Exposing JPA transaction as JDBC [org.springframework
 .orm.jpa.vendor.HibernateJpaDialect$HibernateConnectionHandle@12fdbea]
2020-07-29 12:23:24.754 DEBUG 25992 --- [  restartedMain] o.s.orm.jpa.JpaTransactionManager        : Found thread-bound EntityManager [SessionImpl
 (4619534<open>)] for JPA transaction
2020-07-29 12:23:24.754 DEBUG 25992 --- [  restartedMain] o.s.orm.jpa.JpaTransactionManager        : Participating in existing transaction
2020-07-29 12:23:24.762 DEBUG 25992 --- [  restartedMain] o.s.orm.jpa.JpaTransactionManager        : Found thread-bound EntityManager [SessionImpl
 (4619534<open>)] for JPA transaction
2020-07-29 12:23:24.762 DEBUG 25992 --- [  restartedMain] o.s.orm.jpa.JpaTransactionManager        : Participating in existing transaction
Hibernate: insert into users (user_mail, user_pass) values (?, ?)
2020-07-29 12:23:24.826 DEBUG 25992 --- [  restartedMain] o.s.orm.jpa.JpaTransactionManager        : Found thread-bound EntityManager [SessionImpl
 (4619534<open>)] for JPA transaction
2020-07-29 12:23:24.826 DEBUG 25992 --- [  restartedMain] o.s.orm.jpa.JpaTransactionManager        : Participating in existing transaction
2020-07-29 12:23:24.831 DEBUG 25992 --- [  restartedMain] o.s.orm.jpa.JpaTransactionManager        : Found thread-bound EntityManager [SessionImpl
 (4619534<open>)] for JPA transaction
2020-07-29 12:23:24.831 DEBUG 25992 --- [  restartedMain] o.s.orm.jpa.JpaTransactionManager        : Participating in existing transaction
Hibernate: insert into notes (message, title, user_id) values (?, ?, ?)
2020-07-29 12:23:24.837 DEBUG 25992 --- [  restartedMain] o.s.orm.jpa.JpaTransactionManager        : Initiating transaction commit
2020-07-29 12:23:24.837 DEBUG 25992 --- [  restartedMain] o.s.orm.jpa.JpaTransactionManager        : Committing JPA transaction on EntityManager [SessionImpl
 (4619534<open>)]
2020-07-29 12:23:24.848 DEBUG 25992 --- [  restartedMain] o.s.orm.jpa.JpaTransactionManager        : Closing JPA EntityManager [SessionImpl(4619534<open>)]
 after transaction
```

CASE 2:



In case, if registerUser method is called directly and it has propagation as REQUIRED, then it will create it's own new transaction.

Code: It's the main class, which calls registerUser from User Service.

```
package com.techgeeknext;

import com.techgeeknext.modal.User;
import com.techgeeknext.service.UserNotesLinkService;
import com.techgeeknext.service.UserService;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootAppl
import org.springframework.context.ApplicationContext;
import org.springframework.data.jpa.repository.config.Enable
import org.springframework.web.bind.annotation.RestControlle

@SpringBootApplication
@EnableJpaAuditing
@RestController
public class TransactionManagementApplication {

    public static void main(String[] args) throws Exception
        ApplicationContext context = SpringApplication.run(T
        UserNotesLinkService userNotesLinkService = context.

        UserService userService = context.getBean(UserServic

        /* TEST DATA FOR TESTING PROPAGATION FOR SINGLE SERV
        User user1 = new User();
        user1.setUserMail("newuser@gmail.com");
        user1.setUserPass("23456ghqwe");
        userService.registerUser(user1);
    }
}
```

UserServiceImpl: This is the implementation class for the registerUser.

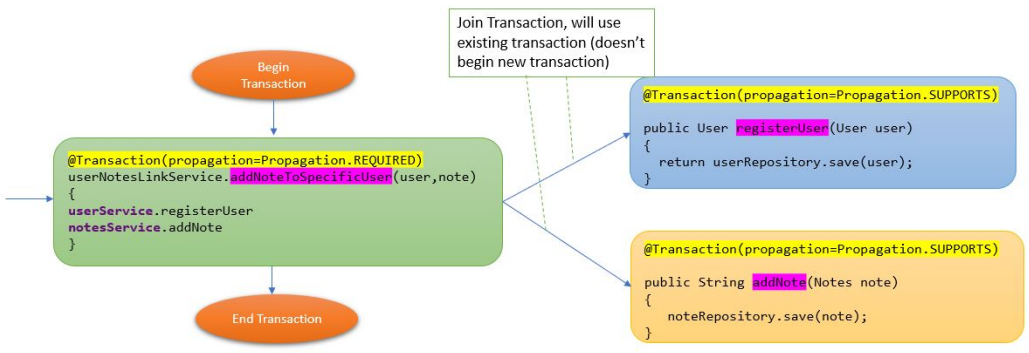
```java
package com.techgeeknext.service.impl;

import com.techgeeknext.modal.User;
import com.techgeeknext.repository.UserRepository;
import com.techgeeknext.service.UserService;
import org.springframework.beans.factory.annotation.Autowire
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Propagatio
import org.springframework.transaction.annotation.Transactio

@Service
public class UserServiceImpl implements  UserService{
    @Autowired
    UserRepository userRepository;


    @Override
    @Transactional(propagation= Propagation.REQUIRED)
    public User registerUser(User user)
    {
        return userRepository.save(user);
    }
}
```

Output:

```
2020-07-29 12:33:54.300  INFO 25936 --- [  restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http) with context path
 ''
2020-07-29 12:33:54.303  INFO 25936 --- [  restartedMain] c.t.TransactionManagementApplication     : Started TransactionManagementApplication in 3.731
 seconds (JVM running for 4.161)
2020-07-29 12:33:54.308 DEBUG 25936 --- [  restartedMain] o.s.orm.jpa.JpaTransactionManager        : Creating new transaction with name [com.techgeeknext
.service.impl.UserServiceImpl.registerUser]: PROPAGATION_REQUIRED ISOLATION_DEFAULT
2020-07-29 12:33:54.308 DEBUG 25936 --- [  restartedMain] o.s.orm.jpa.JpaTransactionManager        : Opened new EntityManager [SessionImpl(17169324<open>)]
 for JPA transaction
2020-07-29 12:33:54.314 DEBUG 25936 --- [  restartedMain] o.s.orm.jpa.JpaTransactionManager        : Exposing JPA transaction as JDBC [org.springframework
.orm.jpa.vendor.HibernateJpaDialect$HibernateConnectionHandle@1c38e05]
2020-07-29 12:33:54.325 DEBUG 25936 --- [  restartedMain] o.s.orm.jpa.JpaTransactionManager        : Found thread-bound EntityManager [SessionImpl
(17169324<open>)] for JPA transaction
2020-07-29 12:33:54.325 DEBUG 25936 --- [  restartedMain] o.s.orm.jpa.JpaTransactionManager        : Participating in existing transaction
Hibernate: insert into users (user_mail, user_pass) values (?, ?)
2020-07-29 12:33:54.385 DEBUG 25936 --- [  restartedMain] o.s.orm.jpa.JpaTransactionManager        : Initiating transaction commit
2020-07-29 12:33:54.385 DEBUG 25936 --- [  restartedMain] o.s.orm.jpa.JpaTransactionManager        : Committing JPA transaction on EntityManager [SessionImpl
(17169324<open>)]
2020-07-29 12:33:54.399 DEBUG 25936 --- [  restartedMain] o.s.orm.jpa.JpaTransactionManager        : Closing JPA EntityManager [SessionImpl(17169324<open>)]
 after transaction
```

2. SUPPORTS Propagation:

CASE 1:

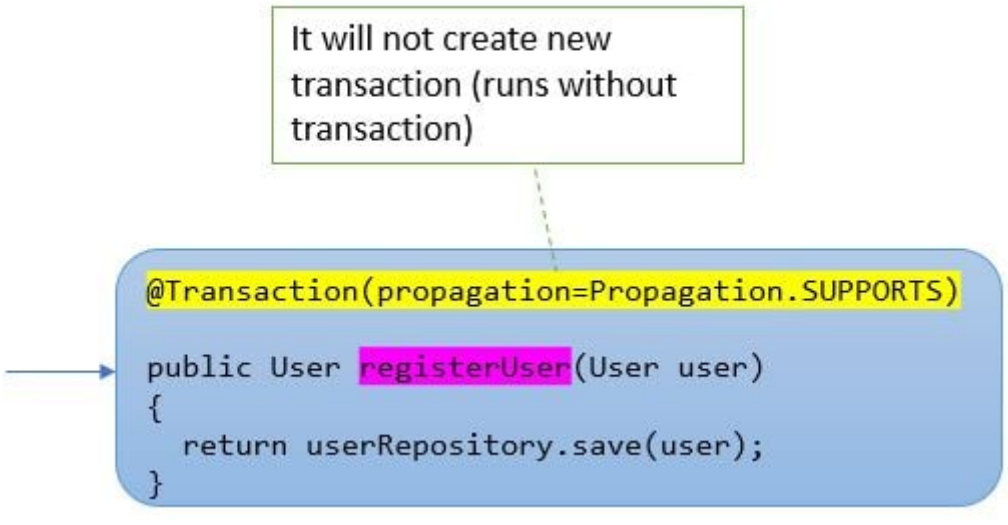


In this case, if calling service addNoteToSpecficUser having RE-QUIRED propagation, and registerUser and addNote having SUP-PORT propagation then registerUser and addNote will make use of existing transaction created from addNoteToSpecficUser.

If the calling service (addNoteToSpecficUser) does not have transaction, the registerUser and addNote with SUPPORT propagation will not creates it's own transaction.

So in case of SUPPORT, registerUser and addNote will be utilizing calling service transaction if it exist, else it'll not create new and run without any transaction.

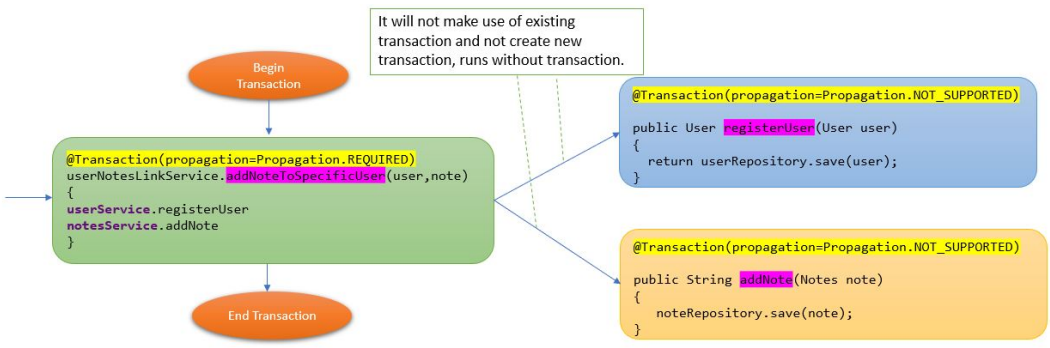
CASE 2:



In case, if registerUser method is called directly and it has propagation as SUPPORT, then it will not create it's own new transaction and run without transaction.

3. NOT_SUPPORTED Propagation:

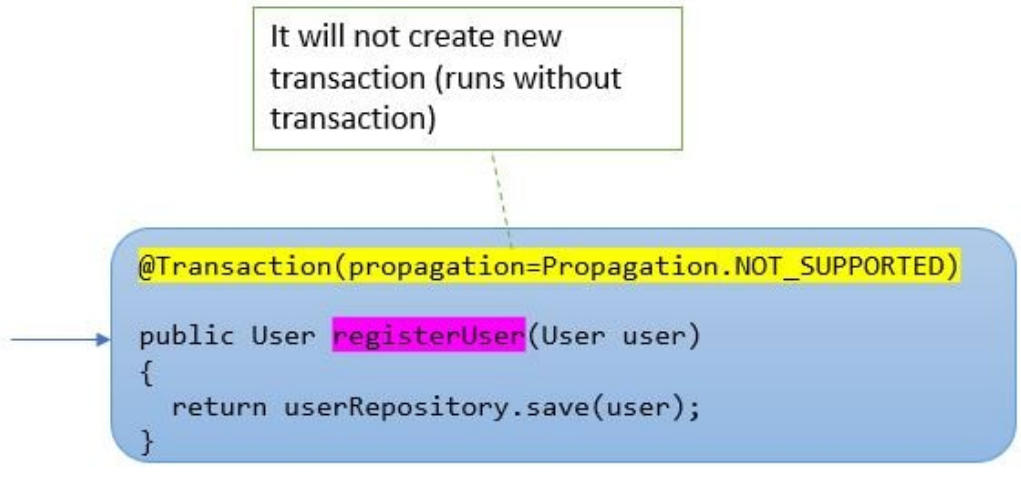
CASE 1:



In this case, if calling service addNoteToSpecficUser having RE-QUIRED propagation, and registerUser and addNote having NOT_SUPPORTED propagation then registerUser and addNote will not make use of existing transaction created from addNoteToSpecficUser and nor it will creates it's own, rather it runs without any transaction.

NOT_SUPPORTED, will always run without transaction and doesn't creates it's own transaction and also not utilizes the call-ing service transaction.

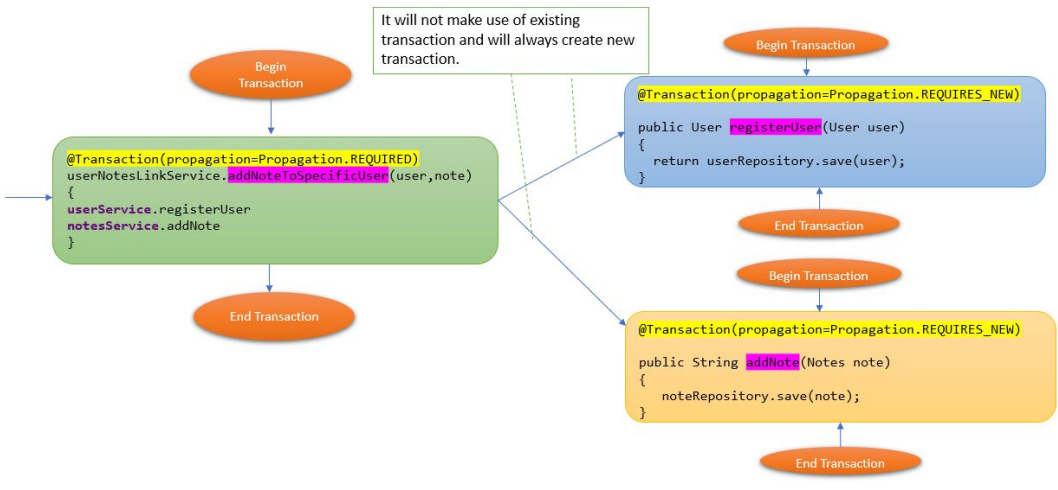
CASE 2:



In case, if registerUser method is called directly and it has propagation as NOT_SUPPORTED, then it will not create it's own new transaction and run without transaction.

4. **REQUIRES_NEW Propagation:**

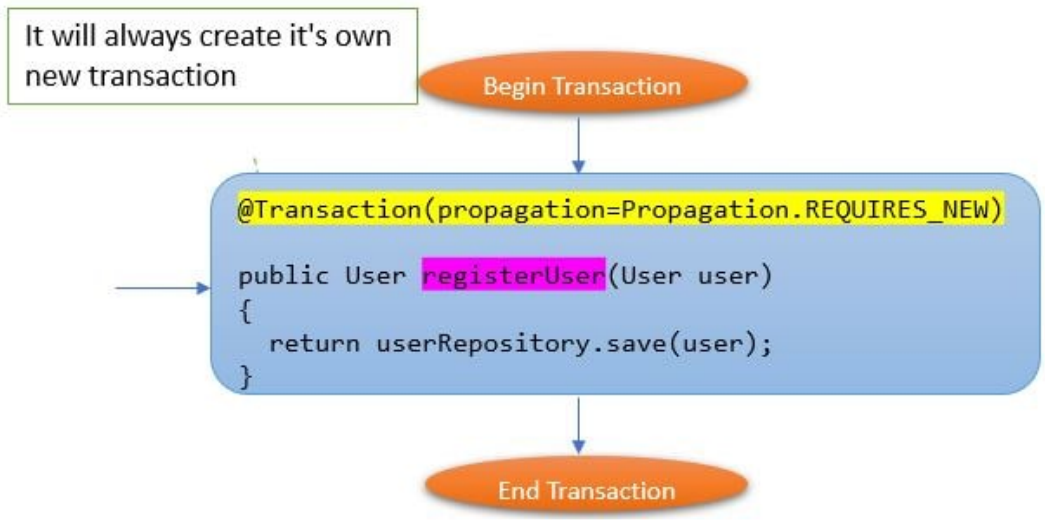
CASE 1:



In this case, if calling service addNoteToSpecficUser having REQUIRED propagation, and registerUser and addNote having REQUIRES_NEW propagation then registerUser and addNote will always creates it's own transaction and doesn't utilizes the existing/calling service transaction.

In short, the service which has REQUIRES_NEW will always creates it's own new transaction and doesn't use it's calling service transaction.

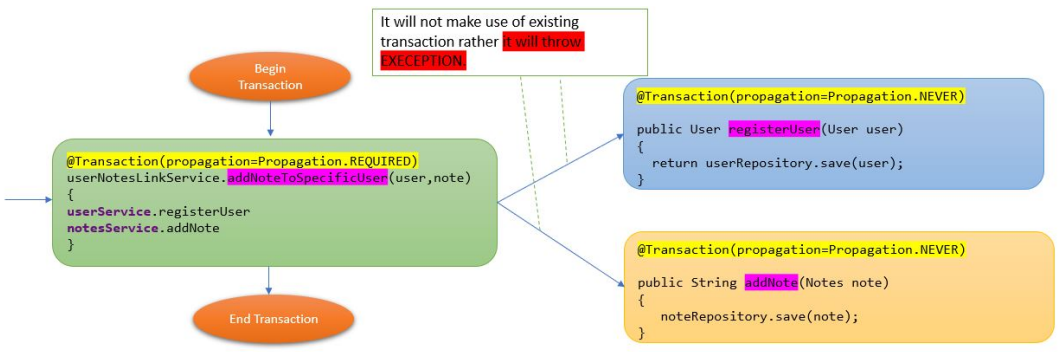
CASE 2:



In case, if registerUser method is called directly and it has propagation as REQUIRES_NEW, then it will always create it's own new transaction.

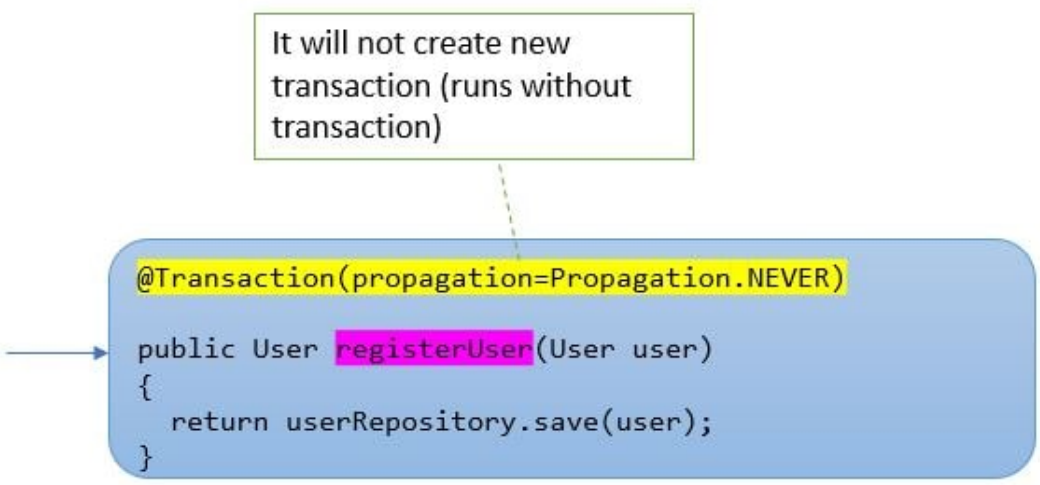
5. **NEVER Propagation:**

CASE 1:



In this case, if calling service addNoteToSpecficUser having RE-QUIRED propagation, and registerUser and addNote having NEVER propagation then registerUser and addNote will not make use of existing transaction rather it will throw EXECEPTION.
And if calling service addNoteToSpecficUser doesn't have any transaction, then registerUser and addNote will not create it's own transaction and it'll run without transaction.

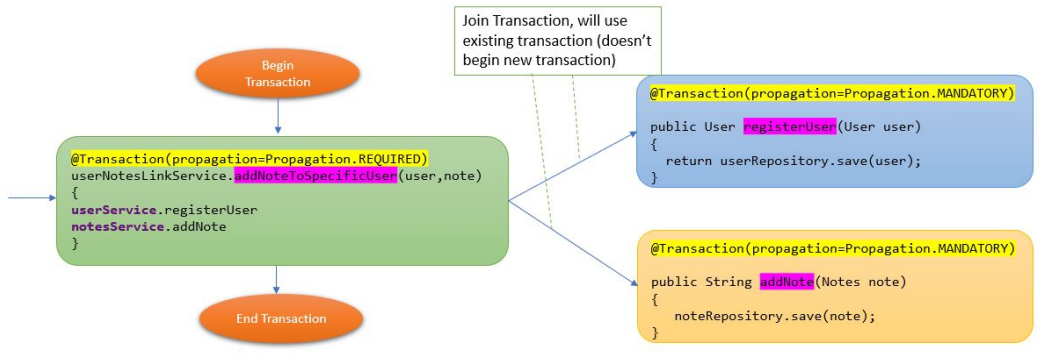
CASE 2:



In case, if registerUser method is called directly and it has propagation as NEVER, then it will NEVER create new transaction and runs without transaction.

6. **MANDATORY Propagation:**

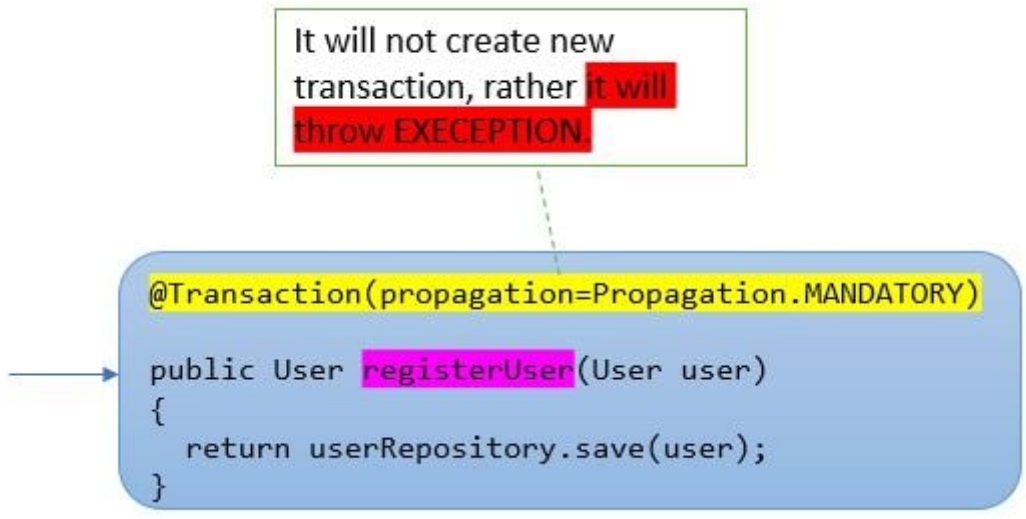
CASE 1:



In this case, if calling service addNoteToSpecficUser having REQUIRED propagation, and registerUser and addNote having MANDATORY propagation then registerUser and addNote will make use of existing transaction.
And if calling service (addNoteToSpecficUser) doesn't have trsation then registerUser and addNote having MANDATORY propagation, will throw EXCEPTION.

So i short, calling service (addNoteToSpecficUser) should transaction else service which calls calling service will throw exception.

CASE 2:



In case, if registerUser method is called directly with MANDATORY propagation, then it will throw an EXCEPTION.