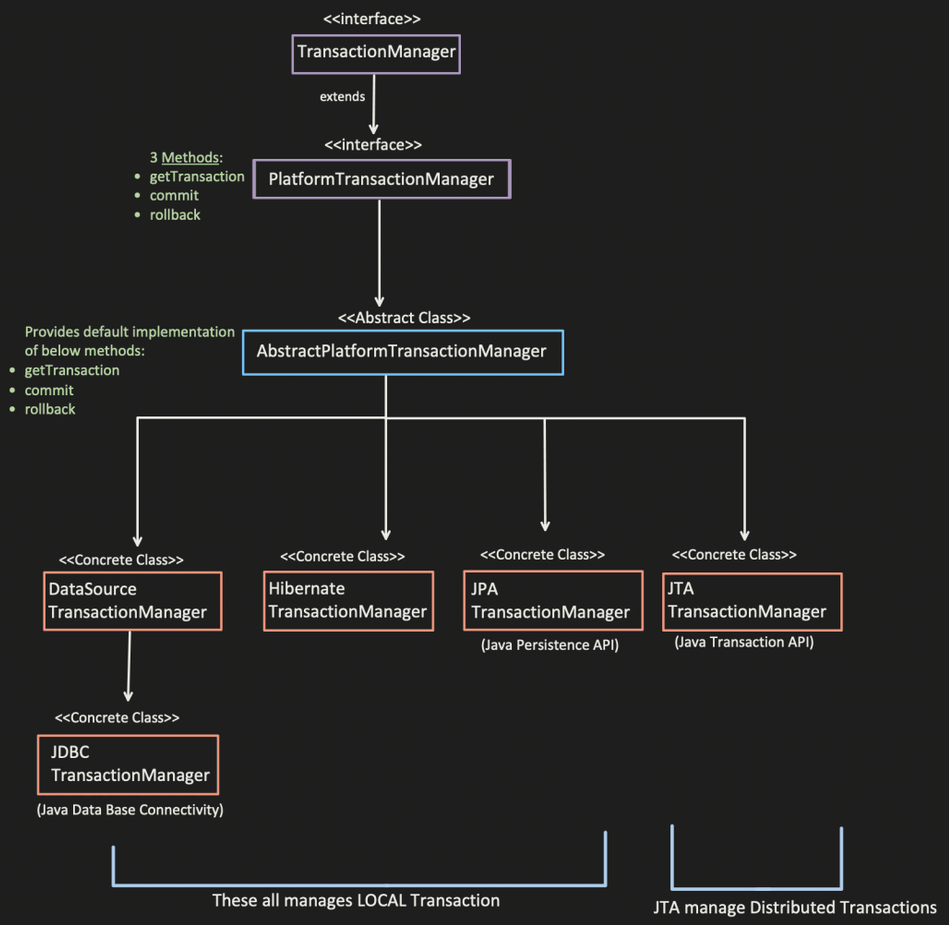
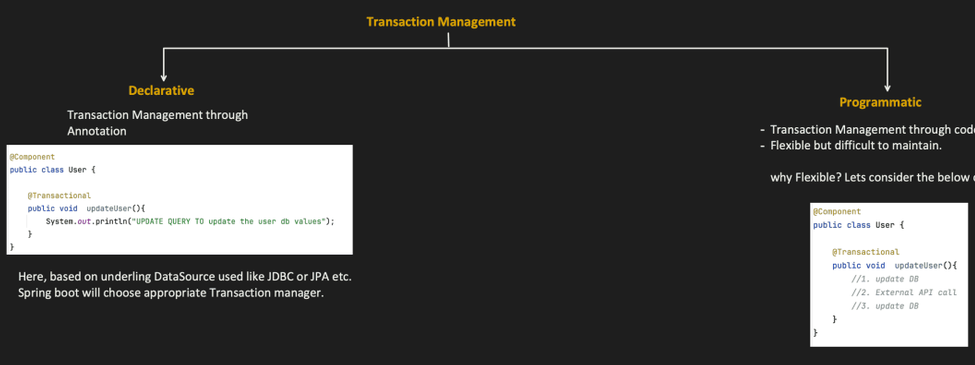
Hierarchy of Transaction Managers





Declarative

Transaction Management through Annotation

```java
@Component
public class User {

    @Transactional
    public void  updateUser(){
        System.out.println("UPDATE QUERY TO update the user db values");
    }
}
```

Here, based on underling DataSource used like JDBC or JPA etc. Spring boot will choose appropriate Transaction manager.

```java
@Configuration
public class AppConfig {

    @Bean
    public DataSource dataSource() {
        DriverManagerDataSource dataSource = new DriverManagerDataSource();
        dataSource.setDriverClassName("org.h2.Driver");
        dataSource.setUrl("jdbc:h2:mem:testdb");
        dataSource.setUsername("sa");
        dataSource.setPassword("");
        return dataSource;
    }

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

```java
@Component
public class UserDeclarative {

    @Transactional(transactionManager = "userTransactionManager")
    public void updateUserProgrammatic() {
        //SOME DB OPERATIONS
        System.out.println("Insert Query ran");
        System.out.println("Update Query ran");
    }
}
```

2 ways to implement by Programmatic

Approach 1

```java
@Configuration
public class AppConfig {

    @Bean
    public DataSource dataSource() {
        DriverManagerDataSource dataSource = new DriverManagerDataSource();
        dataSource.setDriverClassName("org.h2.Driver");
        dataSource.setUrl("jdbc:h2:mem:testdb");
        dataSource.setUsername("sa");
        dataSource.setPassword("");
        return dataSource;
    }

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

```java
@Component
public class UserProgrammaticApproach1 {

    PlatformTransactionManager userTransactionManager;

    UserProgrammaticApproach1(PlatformTransactionManager userTransactionManager) {
        this.userTransactionManager = userTransactionManager;
    }

    public void updateUserProgrammatic(){
        TransactionStatus status = userTransactionManager.getTransaction( definition: null);
        try {
            //SOME INITIAL SET OF DB OPERATIONS
            System.out.println("Insert Query run1");
            System.out.println("Update Query run1");
            userTransactionManager.commit(status);
        }
        catch (Exception e) {
            userTransactionManager.rollback(status);
        }
    }
}
```

Approach 2

Usage of TransactionTemplate

```java
@Configuration
public class AppConfig {

    @Bean
    public DataSource dataSource() {
        DriverManagerDataSource dataSource = new DriverManagerDataSource();
        dataSource.setDriverClassName("org.h2.Driver");
        dataSource.setUrl("jdbc:h2:mem:testdb");
        dataSource.setUsername("sa");
        dataSource.setPassword("");
        return dataSource;
    }

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

    @Bean
    public TransactionTemplate transactionTemplate(PlatformTransactionManager userTransactionManager) {
        return new TransactionTemplate(userTransactionManager);
    }
}
```

```java
@Component
public class UserProgrammaticApproach2 {

    TransactionTemplate transactionTemplate;

    public UserProgrammaticApproach2(TransactionTemplate transactionTemplate){
        this.transactionTemplate = transactionTemplate;
    }

    public void updateUserProgrammatic(){
        TransactionCallback<TransactionStatus> dbOperationsTask = (TransactionStatus status) -> {
            System.out.println("Insert Query ran");
            System.out.println("Update Query ran");
            return status;
        };
        TransactionStatus status = transactionTemplate.execute(dbOperationsTask);
    }

}
```

Now, lets see Propagation

When we try to create a new Transaction, it first check the PROPAGATION value set, and this tell whether we have to create new transaction or not.

- REQUIRED (default propagation):

  @Transactional(propagation=Propagation.*REQUIRED*)
  if(parent txn present)
  Use it;
  else
  Create new transaction;

- REQUIRED_NEW:

  @Transactional(propagation=Propagation.*REQUIRED_NEW*)
  if(parent txn present)
  Suspend the parent txn;
  Create a new Txn and once finished;
  Resume the parent txn;
  else
  Create new transaction and execute the method;

- SUPPORTS:

  @Transactional(propagation=Propagation.*SUPPORTS*)
  if(parent txn present)
  Use it;
  Else
  Execute the method without any transaction;

- NOT_SUPPORTED:

```
@Transactional(propagation=Propagation.NOT_SUPPORTED)
    if(parent txn present)
        Suspend the parent txn;
        Execute the method without any transaction;
        Resume the parent txn;
    else
        Execute the method without any transaction;
```

- MANDATORY:

```
@Transactional(propagation=Propagation.MANDATORY)
    if(parent txn present)
        Use it;
    Else
        Throw exception;
```

- NEVER:

```
@Transactional(propagation=Propagation.MANDATORY)
    if(parent txn present)
        Throw exception;
    Else
        Execute the method without any transaction;
```

Declarative way of usage:

```java
@Component
public class UserDeclarative {

    @Autowired
    UserDAO userDAObj;

    @Transactional
    public void updateUser() {
        System.out.println("Is transaction active: " +  TransactionSynchronizationManager.isActualTransactionActive());
        System.out.println("Current transaction name: " + TransactionSynchronizationManager.getCurrentTransactionName());

        System.out.println("Some initial DB operation");
        userDAObj.dbOperationWithRequiredPropagation();
        System.out.println("Some final DB operation");
    }

    public void updateUserFromNonTransactionalMethod() {
        System.out.println("Is transaction active: " +  TransactionSynchronizationManager.isActualTransactionActive());
        System.out.println("Current transaction name: " + TransactionSynchronizationManager.getCurrentTransactionName());
        System.out.println("Some initial DB operation");
        userDAObj.dbOperationWithRequiredPropagation();
        System.out.println("Some final DB operation");
    }
}
```

```java
@Component
public class UserDAO {

    @Transactional(propagation = Propagation.REQUIRED)
    /**
     * if(parent txn present)
     *       use it
     * else
     *      create new
     *
     */
    public void dbOperationWithRequiredPropagation() {

        //EXECUTE DB QUERIES
        boolean isTransactionActive = TransactionSynchronizationManager.isActualTransactionActive();
        String currentTransactionName = TransactionSynchronizationManager.getCurrentTransactionName();
        System.out.println("*******************************");
        System.out.println("Propagation.REQUIRED: Is transaction active: " + isTransactionActive);
        System.out.println("Propagation.REQUIRED: Current transaction name: " + currentTransactionName);
        System.out.println("*******************************");

    }
}
```

Output:

```
Is transaction active: true
Current transaction name: com.conceptandcoding.learningspringboot.TransactionManagement.UserDeclarative.updateUser
Some initial DB operation
*******************************
Propagation.REQUIRED: Is parent transaction active: true
Propagation.REQUIRED: Current transaction name: com.conceptandcoding.learningspringboot.TransactionManagement.UserDeclarative.updateUser
*******************************
Some final DB operation
```

```
Is transaction active: false
Current transaction name: null
Some initial DB operation
*****************************
Propagation.REQUIRED: Is parent transaction active: true
Propagation.REQUIRED: Current transaction name: com.conceptandcoding.learningspringboot.TransactionManagement.UserDAO.dbOperationWithRequiredPropagation
*****************************
Some final DB operation
```

Programmatic way of usage:

(Approach 1)

```java
@Component
public class UserDeclarative {

    @Autowired
    UserDAO userDAOobj;

    @Transactional
    public void updateUser() {
        System.out.println("Is transaction active: " +  TransactionSynchronizationManager.isActualTransactionActive());
        System.out.println("Current transaction name: " + TransactionSynchronizationManager.getCurrentTransactionName());

        System.out.println("Some initial DB operation");
        userDAOobj.dbOperationWithRequiredPropagationUsingProgrammaticApproach1();
        System.out.println("Some final DB operation");
    }

    public void updateUserFromNonTransactionalMethod() {
        System.out.println("Is transaction active: " +  TransactionSynchronizationManager.isActualTransactionActive());
        System.out.println("Current transaction name: " + TransactionSynchronizationManager.getCurrentTransactionName());
        System.out.println("Some initial DB operation");
        userDAOobj.dbOperationWithRequiredPropagationUsingProgrammaticApproach1();
        System.out.println("Some final DB operation");
    }
}
```

```java
@Component
public class UserDAO {

    PlatformTransactionManager userTransactionManager;

    UserDAO(PlatformTransactionManager userTransactionManager) {
        this.userTransactionManager = userTransactionManager;
    }

    /**
     * if(parent txn present)
     *        use it
     * else
     *      create new
     */
    public void dbOperationWithRequiredPropagationUsingProgrammaticApproach1(){
        DefaultTransactionDefinition transactionDefinition = new DefaultTransactionDefinition();
        transactionDefinition.setName("Testing REQUIRED propagation");
        transactionDefinition.setPropagationBehavior(TransactionDefinition.PROPAGATION_REQUIRED);
        TransactionStatus status = userTransactionManager.getTransaction(transactionDefinition);
        try {
            //EXECUTE operation
            System.out.println("******************************");
            System.out.println("Propagation.REQUIRED: Is  transaction active: " + TransactionSynchronizationManager.isActualTransactionActive());
            System.out.println("Propagation.REQUIRED: Current transaction name: " + TransactionSynchronizationManager.getCurrentTransactionName());
            System.out.println("******************************");
            userTransactionManager.commit(status);
        }
        catch (Exception e) {
            userTransactionManager.rollback(status);
        }
    }
```

(Approach 2) : using TransactionTemplate

```java
@Configuration
public class AppConfig {

    @Bean
    public DataSource dataSource() {
        DriverManagerDataSource dataSource = new DriverManagerDataSource();
        dataSource.setDriverClassName("org.h2.Driver");
        dataSource.setUrl("jdbc:h2:mem:testdb");
        dataSource.setUsername("sa2");
        dataSource.setPassword("");
        return dataSource;
    }

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

    @Bean
    public TransactionTemplate transactionTemplate(PlatformTransactionManager userTransactionManager) {
        TransactionTemplate transactionTemplate = new TransactionTemplate(userTransactionManager);
        transactionTemplate.setPropagationBehavior(TransactionDefinition.PROPAGATION_REQUIRED);
        transactionTemplate.setName("TRANSACTION TEMPLATE REQUIRED PROPAGATION");
        return transactionTemplate;
    }
}
```

```java
@Component
public class UserService {

    @Autowired
    UserDAO userDAOobj;

    @Transactional
    public void updateUser() {
        System.out.println("Is transaction active: " +  TransactionSynchronizationManager.isActualTransactionActive());
        System.out.println("Current transaction name: " + TransactionSynchronizationManager.getCurrentTransactionName());

        System.out.println("Some initial DB operation");
        userDAOobj.dbOperationWithRequiredPropagationUsingProgrammaticApproach2();
        System.out.println("Some final DB operation");
    }

    public void updateUserFromNonTransactionalMethod() {
        System.out.println("Is transaction active: " +  TransactionSynchronizationManager.isActualTransactionActive());
        System.out.println("Current transaction name: " + TransactionSynchronizationManager.getCurrentTransactionName());
        System.out.println("Some initial DB operation");
        userDAOobj.dbOperationWithRequiredPropagationUsingProgrammaticApproach2();
        System.out.println("Some final DB operation");
    }
}
```

```java
@Component
public class UserDAO {

    TransactionTemplate transactionTemplate;

    UserDAO(TransactionTemplate transactionTemplate) {
        this.transactionTemplate = transactionTemplate;
    }

    /**
     * if(parent txn present)
     * use it
     * else
     * create new
     */
    public void dbOperationWithRequiredPropagationUsingProgrammaticApproach2() {

        TransactionCallback<TransactionStatus> operations = (TransactionStatus status) -> {
            //some operations for this method
            System.out.println("*******************************");
            System.out.println("Propagation.REQUIRED: Is  transaction active: " + TransactionSynchronizationManager.isActualTransactionActive());
            System.out.println("Propagation.REQUIRED: Current transaction name: " + TransactionSynchronizationManager.getCurrentTransactionName());
            System.out.println("*******************************");
            return status;
        };

        TransactionStatus status = transactionTemplate.execute(operations);
    }
}
```

Output:

```
Is transaction active: true
Current transaction name: com.conceptandcoding.learningspringboot.TransactionManagement.UserDeclarative.updateUser
Some initial DB operation
*******************************
Propagation.REQUIRED: Is  transaction active: true
Propagation.REQUIRED: Current transaction name: com.conceptandcoding.learningspringboot.TransactionManagement.UserDeclarative.updateUser
*******************************
Some final DB operation
```

```
Is transaction active: false
Current transaction name: null
Some initial DB operation
*******************************
Propagation.REQUIRED: Is  transaction active: true
Propagation.REQUIRED: Current transaction name: TRANSACTION TEMPLATE REQUIRED PROPAGATION
*******************************
Some final DB operation
```