@Transactional

* Spring boot uses Transaction Manager to create new transaction of join an existing one.
* Manages the transaction life cycle.
* Transaction Manger supports different transaction isolation levels
* Isolation levels
  + READ\_COMMITED
  + READ\_UNCOMMMITED
  + REPEATABLE\_READ
  + SERIALIZABLE

@Transactional

Creates transaction if none exists or joins existing transaction if one is already active

@Transactional (propagation.REQUIRES\_NEW) – creates new transaction suspending the current transaction.

@Transactional  
 public void methodA() {  
 // ... some code here  
 methodB();  
 // ... some code here  
 }  
  
 @Transactional(propagation = Propagation.REQUIRES\_NEW)  
 public void methodB() {  
 // ... some code here  
 }

Spring uses proxy based approach to manage transactions

If within the class the call is made to original instance(not proxy) and the transactional behaviour is not applied.

Aspect J based weaving or move the transactional method to separate class.

* Spring cretes new proxy around the target bean and applies the transactional behaviour based on propagation setting of calling method
* @Transactional  
   public void methodA() {  
   // ... some code here  
   methodB();  
   // ... some code here  
   }  
    
   @Transactional(propagation = Propagation.REQUIRES\_NEW)  
   public void methodB() {  
   // ... some code here  
   }

No rollbak in case of unchecked exception

* @Trasnacional – only works for public method
* Private methods are not visible to proxy and cannot be wrapped in transactional context.
* Make the @Trasnactional to different class that calls bot method and method b

How @Trasnactional Annotation handles concurrency issues ?

* It handles the concurrency issues by serializing trasnactions that modify the same data, preventing multiple threads from modifyinh samed data aat same time