# Chapter 12. Transaction Management

## 12.1. Introduction

In this lab, you will learn to configure and test GemFire transactions. You can group cache updates with GemFire transactions. Transactions allow you to create dependencies between cache modifications so they all either complete together or fail together.

**Concepts you will gain experience with:**

- Creating the `Customer` region, co-located with the `Order` region.
- Create a Java class to implement GemFire transaction by using the `CacheTransactionManager` API.
- Use transactions to update data on both the `Customer` region and the `Order` region.

**Estimated completion time**: 45 minutes

## 12.2. Quick Instructions

Quick instructions for this exercise have been embedded within the lab materials in the form of TODO comments. To display them, open the `Tasks` view (Window -> Show view -> Tasks (*not Task List*)).

## 12.3. Detailed Instructions

Instructions for this lab are divided into specific sections. Each section describes the steps to perform specific tasks. Open the `transactions` project in the STS.

### 12.3.1. Implement the TransactionService functionality

(`TODO-01`): Before beginning, open the test harness, `TransactionalTest` under the `com.gopivotal.bookshop.tests` under `src/test/java`. Observe the two tests that will be executed to validate a correct transaction commit and a transaction rollback. Note that the first test asserts that the transaction commits successfully and that the associated values have been updated.

Open the `TransactionsService` class within `com.gopivotal.bookshop.buslogic` package and locate the `updateCustomerAndOrder()` method. Your responsibility is to write the necessary code to perform the fetch and update within a transaction. Perform the following steps.

1. (`TODO-02`): Define `CacheTransactionManager`, and get the transaction context.
2. Add the transactional code.
   a. (`TODO-03a`) Begin the transaction.
   b. (`TODO-03b`) Retrieve and update the `Customer` using the customerKey and updatedCustomerPhone values. Similarly, retrieve and update the Order using the orderKey and updatedOrderDate value.
   c. (`TODO-03c`): Save the `Customer` and `Order` objects in the cache with the same keys.
   d. (`TODO-03d`): Commit the transaction.
3. (`TODO-04`) Add code in the exception handler to roll back the transaction in case of any error with the transaction. This handler should catch the exception, roll back the transaction and re-throw the exception so the test harness can catch it.

### 12.3.2. Cache configuration and Server Start

There are several configuration changes you'll want to make to ensure transactions between two partitioned regions work properly.

1. Open the `serverCache.xml` file.
2. (`TODO-05`): Set the cache configuration attribute `copy-on-read` to `true` to avoid in place changes.
3. (`TODO-06`): Add necessary configuration so that the `Order` region is co-located with the `Customer` region.
4. (`TODO-07`): Save your work and then start the locator and two servers using the start script in the project folder.

   **Tip**

   From the command prompt, be sure to first run either **source gf.config** or **setEnv.bat**, which are both in the `server-bootstrap/scripts` project. Next, run **gfsh run --file=serverStart.gf**

### 12.3.3. Running the tests

Run the tests and verify both tests pass.

Also, take a look at the `listener.log` file that is created in the transactions folder. The last entries should look something like the following output.

```
[info 2015/11/18 13:27:43.635 MST server2 <ServerConnection on port 56069 Thread 0> tid=0x4a] afterUpdate:   Entry updated for key: 1001
                 Old value: Customer [customerNumber=C001, firstName=Lula, lastName=Wax, Phone=123-654-543, Address=Address [addressTag=HOME, addressLine1=123 Main St., city=
                 New Value: Customer [customerNumber=C001, firstName=Lula, lastName=Wax, Phone=222-22222-0000, Address=Address [addressTag=HOME, addressLine1=123 Main St., ci

[info 2015/11/18 13:27:43.639 MST server2 <ServerConnection on port 56069 Thread 0> tid=0x4a] afterUpdate:   Entry updated for key: 1001
                 Old value: Order [orderNumber=ORD001, orderDate=Tue Dec 03 00:00:00 MST 2013, customerNumber=C001, totalPrice=103.5, orderItems=[ProductItem:  [itemNumber=P0
                 New Value: Order [orderNumber=ORD001, orderDate=Thu Apr 25 00:00:00 MDT 2013, customerNumber=C001, totalPrice=103.5, orderItems=[ProductItem:  [itemNumber=P0
```

What this is actually showing is that the updates to the entries once the transaction commits. These are captured and logged using the `LoggingCacheListener` to fire on the `afterUpdate()` event.

### 12.3.4. Implementing a TransactionListener

In this next section, you will implement a `TransactionListener` that will allow you to also track the transactional events, namely the commit and rollback operations. This will allow you to verify that the commit and rollback events actually occur.

To begin, open the `LoggingTransactionListener` class. Note that this class extends `TransactionListenerAdapter` as well as implementing `Declarable`. You will implement two methods that override the methods on the adapter class.

1. (`TODO-08`): First, add an `afterCommit()` method that overrides the one in TransactionListenerAdapter. In the body of the method, use the logger to log an INFO message that this is an afterCommit event and then also use the `TransactionEvent` object that is passed in to obtain the list of related cache events. Iterate over these events, printing out the event details.

   **Tip**

   Refer to one of the methods of the `LoggingCacheListener` for an example of what you might want to print for each event.

2. (`TODO-09`): Similarly, add an `afterRollback()` method that overrides the `TransactionListenerAdapter` method and use it to log an `INFO` message that this is an `afterRollback` event. If desired, you may also be interested to obtain and print out related operation(s) that were involved in the transaction that rolled back.
3. (`TODO-10`): Open the `serverCache.xml` file and add appropriate configuration to register this LoggingTransactionListener.
4. (`TODO-11`): Use the stop script (`serverStop.gf`) to stop the locator and servers. Then, restart and re-run the TransactionTests test harness.

This time when you look at the end of the `listener.log` file that is created in the `transactions` folder, you should see additional log data that should look something like the following. In this case, you'll notice that the updates happen before the `afterCommit` event is triggered. Notice also that there are two events that are associated with this commit as shown by the event for `Customer` and `Order`.

```
[info 2015/11/18 15:28:03.590 MST server2 <ServerConnection on port 57651 Thread 0> tid=0x4a] afterUpdate:   Entry updated for key: 1001
                 Old value: Customer [customerNumber=C001, firstName=Lula, lastName=Wax, Phone=123-654-543, Address=Address [addressTag=HOME, addressLine1=123 Main St., city=
                 New Value: Customer [customerNumber=C001, firstName=Lula, lastName=Wax, Phone=222-22222-0000, Address=Address [addressTag=HOME, addressLine1=123 Main St., ci

[info 2015/11/18 15:28:03.595 MST server2 <ServerConnection on port 57651 Thread 0> tid=0x4a] afterUpdate:   Entry updated for key: 1001
                 Old value: Order [orderNumber=ORD001, orderDate=Tue Dec 03 00:00:00 MST 2013, customerNumber=C001, totalPrice=103.5, orderItems=[ProductItem:  [itemNumber=P0
                 New Value: Order [orderNumber=ORD001, orderDate=Thu Apr 25 00:00:00 MDT 2013, customerNumber=C001, totalPrice=103.5, orderItems=[ProductItem:  [itemNumber=P0

[info 2015/11/18 15:28:03.597 MST server2 <ServerConnection on port 57651 Thread 0> tid=0x4a] afterCommit: TxId= TXId: 192.168.0.60(DataOperations Client:31819:loner):57660:0

[info 2015/11/18 15:28:03.597 MST server2 <ServerConnection on port 57651 Thread 0> tid=0x4a]    Entry updated for key: 1001
            Old value: Customer [customerNumber=C001, firstName=Lula, lastName=Wax, Phone=123-654-543, Address=Address [addressTag=HOME, addressLine1=123 Main St., city=Topeka
            New Value: Customer [customerNumber=C001, firstName=Lula, lastName=Wax, Phone=222-22222-0000, Address=Address [addressTag=HOME, addressLine1=123 Main St., city=Top

[info 2015/11/18 15:28:03.598 MST server2 <ServerConnection on port 57651 Thread 0> tid=0x4a]    Entry updated for key: 1001
            Old value: Order [orderNumber=ORD001, orderDate=Tue Dec 03 00:00:00 MST 2013, customerNumber=C001, totalPrice=103.5, orderItems=[ProductItem:  [itemNumber=P001, D
            New Value: Order [orderNumber=ORD001, orderDate=Thu Apr 25 00:00:00 MDT 2013, customerNumber=C001, totalPrice=103.5, orderItems=[ProductItem:  [itemNumber=P001, D
```

```
[info 2015/11/18 15:28:03.704 MST server2 <ServerConnection on port 57651 Thread 0> tid=0x4a] afterRollback: TxId= TXId: 192.168.0.60(DataOperations Client:31819:loner):57660

[info 2015/11/18 15:28:03.704 MST server2 <ServerConnection on port 57651 Thread 0> tid=0x4a]    Cache event received with operation: UPDATE
```

Congratulations!! You have completed this lab. Be sure to use the `serverStop.gf` script to stop the locator and servers.