Implementing Your ATG Commerce Solution Rel 10.1

Activity Guide

D79786GC10

Edition 1.0

March 2013

D81007

ORACLE

Authors

Karin Layher

Kerwin Moy

Technical Contributor and Reviewer

Rick Wilson

Editors

Rashmi Rajagopal

Malavika Jinka

Graphic Designer

Maheshwari Krishnamurthy

Publishers

Sujatha Nagendra

Jayanthy Keshavamurthy

Table of Contents

Practices for Lesson 1: Introduction

Chapter 1

Practices for Lesson 1

Practices Overview

There are no practices for this lesson.

Practices for Lesson 2: Commerce Reference Store

Chapter 2

Practice 2-1: Lab Environment Overview

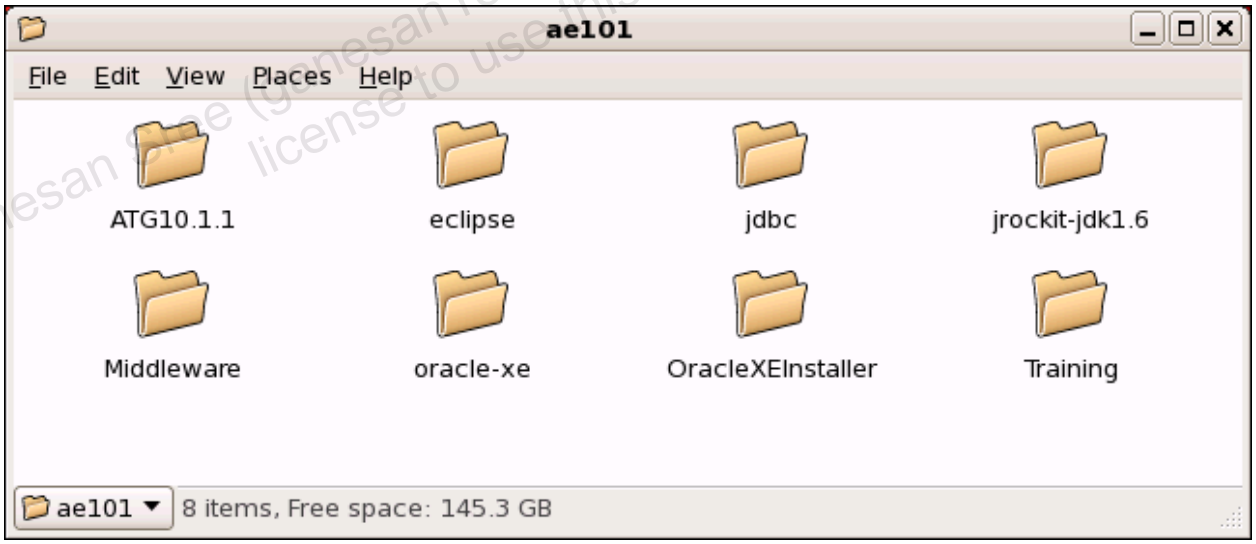
Overview

In this practice, you will become familiar with the lab environment. Your instructor and/or IT department will provide you with an environment on which the following software has been installed:

- ATG 10.1.1 (ATG Platform—all modules)
- ATG Commerce Reference Store 10.1.1
- Oracle 11*g* XE Database
- Weblogic Application Server 10.3.6
- Eclipse 3.7.2 (Indigo) with ATG plug-in
- JRockit JDK 1.6
- Course specific files for student exercises (“COM”)
- A Web browser

Task: Understanding the file structure

Open a terminal window or a file browser and navigate into the `/ae101` folder. Under that folder, you will find subfolders for ATG, Eclipse, Weblogic, Oracle, and Training. Familiarize yourself with the content of these folders as they will be used throughout this course.



Practice 2-2: Getting Started

Overview

In this practice, you will learn how use your development environment. At the end of this exercise you will be able to:

- Start the Weblogic Admin server
- Start the production server
- Start the CA server
- Import the **`MyStore.Custom`** module into Eclipse
- Assemble and deploy the EAR to Weblogic
- Set up directory link for JSPs

Problem Statement

The development environment has already been configured for you. A module called **`MyStore.Custom`** has been created following the steps shown in the slides. In this lab, you will complete the setup by importing the module into Eclipse, learn how to build your module, assemble the EAR, and deploy it to your servers by using CIM. You will also launch each server.

Tasks

Task 1: Starting the production and publishing servers

1. Open the **Comm_shortcuts** folder on your desktop, and then navigate to the **servers** folder.
2. Double-click the **WLSAdmin.sh** icon.
3. Select "Run in Terminal."
4. In the terminal window menu, select **Terminal > Set Title** to enter a title for this window. Enter `"WLS Admin"` and click **Close** to close the pop-up window. Minimize the window.
5. Double-click the **ATGProduction.sh** icon.
6. Select "Run in Terminal."
7. When WebLogic starts up in the terminal window, enter "`weblogic/password1`" as the username and password at the prompts.
8. The server will take several minutes to start. When the startup is complete, you should see a message similar to this:

```
<Dec 5, 2012 2:52:05 PM EST> <Notice> <WebLogicServer> <BEA-
000365> <Server state changed to RUNNING>
<Dec 5, 2012 2:52:05 PM EST> <Notice> <WebLogicServer> <BEA-
000360> <Server started in RUNNING mode>
```

9. When the startup is complete, set the title for this window to `"ATG Production"` as you did in step 4 and proceed to the next step. You may minimize the terminal windows, but **do not close them**, as this stops the servers.
10. Go back to the **Comm_shortcuts** folder and double-click the **Visit Production Store** shortcut to open the storefront in a browser.
11. If there are any errors on the server console, or the store web site did not display correctly, alert your instructor. If there are no errors, take a moment to familiarize yourself with the storefront and the console window.
12. Repeat steps 5-11 for the publishing server with the **ATGPublishing.sh** and **Visit BCC** shortcuts. In this lab, you will only bring up the login page. You will log into the BCC in later lessons.
13. **Note:** During the startup of the publishing server, you will see an error about the `DeploymentFulfiller` unable to acquire a lock similar to the following message. It is specific to how this server is configured and the component startup order. In fact, after the server starts up, the `DeploymentFulfiller` will succeed in getting a lock on the next attempt.

```
**** Error      Thu Dec 27 08:50:41 EST 2012    1356616241633
/atg/epub/deployment/DeploymentFulfiller        ---
java.lang.Exception: Error acquiring lock. Check your client and
server lock manager configurations.
```

A note on building, assembling and deploying code: Throughout this course, you will be asked to build, assemble, and deploy your code to WebLogic. The only server that must remain up is the WebLogic Admin server. The steps you will take are: 1) Build your project in Eclipse, 2) Assemble your EAR in CIM, and 3) deploy the EAR to WebLogic by using the WebLogic Admin UI. After WebLogic deploys the EAR, if that server is already running, WebLogic will restart the server automatically. If it is not running, restart the server by using the shortcuts provided. To bring down a server, type **Ctrl-c** within the terminal window or close the window itself.

Task 2: Importing `MyStore.Custom` into Eclipse

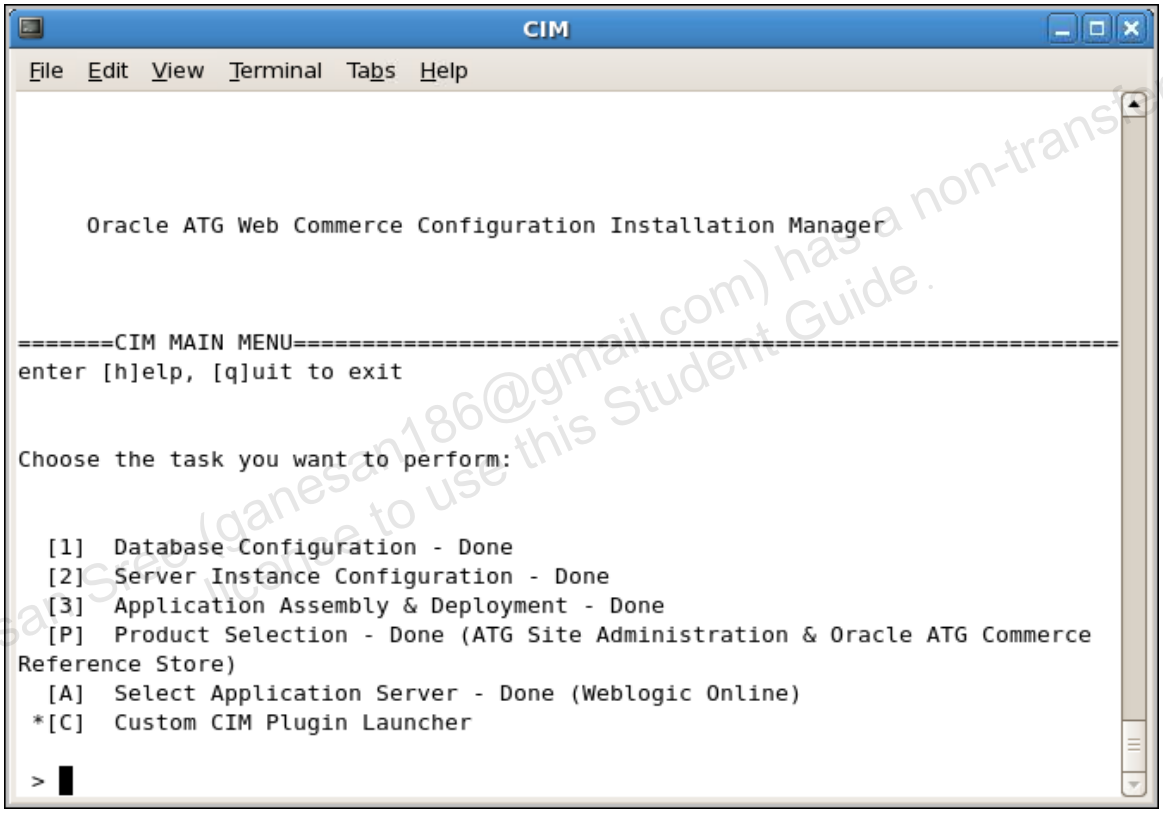
14. Open the `/ae101/eclipse` folder and double-click the **eclipse** icon.
15. Click **ok** to select `/ae101/eclipse/workspace` as your workspace.
16. Click **File > New > Other** or press **Ctrl-N**.
17. Type `atg` in the Wizards type-ahead box.
18. Select **Existing ATG Module** from the list of matching wizards and click **Next**.
19. Type `MyStore.Custom` in the **Project name** field and click Next.

20. Click **Finish** in the pop up window. The `MyStore.Custom` project should now appear in the Package Explorer.
21. Click the Open Perspective icon in the upper right-hand corner ( ) and select **Other**. In the Open Perspective dialog box, select **ATG** and click **OK**.
22. On the left, select the **ATG Component Browser** tab. Verify that `MyStore.Custom` is available on this tab.

Note: Throughout the course, you may see warnings in the eclipse error log about “NLS unused messages”. These do not impact the labs and may be ignored. You may want to periodically clear out the error messages.

Task 3: Assemble and deploy application EAR to Weblogic

23. On the desktop, double-click **Terminal** to open a terminal window.
24. Set the title of this terminal window to `"CIM"`.
25. Change the directory to the **ATG bin directory** by typing the following: `"cd /ae101/ATG10.1.1/home/bin"`. **Tip:** The tab key in the bash shell completes a word for you.
26. Launch the **ATG Configuration InstallationManager (CIM)** by typing `"./cim.sh"`. This tool is used to configure your ATG installation. You will only use it to assemble and deploy your new code to Weblogic.The following screenshot shows the main menu screen for CIM.



Note: You will see the following error during CIM startup: `"mkdir: cannot create directory `/ae101/ATG10.1.1/home/CIM`: File exists"`. This directory has already been created on the initial run and the error can be ignored.

27. Select option **3** `Application Assembly & Deployment` followed by option **A** to select the `atg_production` server instance.
28. Select the default EAR filename `atg_production.ear` by pressing Enter.
29. Select option **D** to deploy the Production EAR file `atg_production.ear` to WebLogic online. Wait a few seconds for the menu to return. When CIM completes, you will see a message similar to:

```
>> File written to
/ae101/ATG10.1.1/home/../home/servers/atg_production/
startServerOnWeblogic.sh
```

30. Select option **O** to configure another server instance and repeat steps 27-29 for the `atg_publishing_lockserver` instance. Minimize the **CIM** terminal window.
31. Double-click the **Visit WLS Admin** icon in **Comm_Shortcuts** to open the WebLogic Admin UI in a browser.
32. Log in to the WebLogic Admin UI with user/password as `weblogic/password1`. This is the application that you use to update the EAR file in WebLogic.
33. In the left navigation menu, click **Deployments**. The right pane refreshes with two EAR files: one for `atg_production` (the storefront) and one for `atg_publishing_lockserver` (CA).
34. Check `atg_production` and click **update** followed by **finish** to select the default source and deployment paths. This will trigger the EAR to be deployed from ATG into WebLogic. Wait a few minutes for the screen to refresh. After updating `atg_production`, repeat the same steps to update `atg_publishing_lockserver`.
35. If your server is already running, after the deployment finishes, the server will be restarted. You may switch to the console window to watch for startup errors. If it is not running, restart the server now.

Task 4: Linking the JSP directories

36. Open a terminal window and rename **store.war** to **store.war.bak** by typing the following:

```
"cd /ae101/ATG10.1.1/MyStore/Storefront/j2ee-apps/Storefront"

"mv store.war store.war.bak"
```

37. Create a link to the production `store.war` directory by executing the following commands:

```
"cd /ae101/ATG10.1.1/MyStore/Storefront/j2ee-apps/Storefront"

"ln -s
/ae101/ATG10.1.1/CommerceReferenceStore/Store/Storefront/j2ee-
apps/Storefront/store.war/"
```

38. The `store.war` folder now exists again, but this time as a link to the corresponding folder under the **Commerce Reference Store** module that will be built into the EAR file we are deploying to WebLogic. With this arrangement, it is important to create a backup of the file we are editing to preserve the original file and get back to the original state if necessary.
39. You may minimize this window for future use.

Bookmark ATG Documentation in Your Browser

40. Visit the documentation page in the browser through the following link: `http://docs.oracle.com/cd/E35318_02/index.html`
41. Click the **View** link next to **Oracle ATG Web Commerce** in the **Product Resources** panel on the left (this will open a new browser tab).
42. Scroll down and locate the **Commerce Programming Guide**. Click the **HTML** link next to the guide name to load this document. Bookmark this link for future reference. You may also use the **PDF** link to store a copy of the guide locally if you prefer.
43. Repeat the previous step for the **Commerce Guide to Setting Up a Store**. These are the manuals most closely tied to the concepts covered in this training course, but you may want to explore other manuals as well.

Practices for Lesson 3: Extending Personalization for Commerce

Chapter 3

Practice 3-1: Extending the User Profile

Overview

In this practice, you will extend the user profile. At the end of this exercise you will be able to:

- Add custom properties to the user profile
- Use `StartSQLRepository` to test changes

Problem Statement

The store needs to extend the user profile in order to store a user's current point balance. For now, this value will be stored in an integer property. Later in the course, this property will be modified to point to a more complex object in the Claimable Repository, so that points can be used as a payment method.

The store will also be implementing ship-to-store functionality, which will allow shoppers to ship items for free to the store they select as their home store. The stores have already been configured as repository items in the Commerce Reference Store, in a repository called Store Repository.

Planning

Take a moment before looking at the next page to think about what you will need to do to accomplish the tasks discussed above. Try to list the extensions and customizations you will need to make in as much detail as possible, including pseudo-code. Refer back to the slides if needed.

After you are done, check your plan against the completed plan on the next page. The completed plan includes suggested names for tables, classes, and properties.

Plan

Task: Extending the user profile

- Create a new table named `mys_user` in the production and publishing datasources for custom properties to be added to profile.
  - The new table will have three columns: `user_id` (`varchar2(40)`), `points` (number(10)), and `store_id` (`varchar2(40)`).
  - **Note:** For this course, all database tables have been created for you. Step 1 in the guided instructions section describes how to view the tables.

- Extend `userProfile.xml` under the **`MyStore.Custom`** `config` to model new properties.
  - Item descriptor to extend is `user`.
  - Table is auxiliary type.
  - `points` is an `int` property with a default value of `0`.
  - `mystore` is an item type property pointing to store in `/atg/store/stores/StoreRepository`.

- Testing:
  - Use `startSQLRepository` to test Profile Repository extensions (go to Step 4 in the Guided Instruction section for more information on how to run this test).
  - Build your module code in Eclipse, assemble and install the new EAR by using CIM, and deploy the EAR by using WebLogic Admin Server.
  - If it is not already running, start the production server and watch for errors.
  - Use ACC to verify that properties are visible and editable.
    - During this test, change the country of the selected store from `USA` to `US` (this is for a later lab).

You are encouraged to try to complete this exercise using only the plan above. You will not need to create the database tables, as they have already been created for you. Use `SQL*Plus` and log into the database schemas for both production and publishing schemas to view the database table `mys_user` (go to Step 1 in the Guided Instruction section for more information on how to use `SQL*Plus`). You should use the suggested variable, class, and component names.

Alternatively, there are step-by-step instructions starting on the next page if you prefer. Whichever method you choose, there are solutions available for reference in the `/ae101/Training/COM/solutions` for each practice should you need them.

A note on testing changes to repository definitions: We will use the `startSQLRepository` utility to test our changes to the Profile Repository definition (and also other repositories in later chapters). This utility attempts to start the repository as if it were being started by a full ATG server. If there are errors, it indicates an issue with the XML definition (for example, the bracket is not closed, or a column or table name is incorrect).

However, `startSQLRepository` does not have access to the WebLogic database connections. It uses its own, through a component called `FakeXADatasource`. The alternate connection to the production database has been configured in a module called **MyStoreFakeXA**. This module's manifest points to **MyStore.Storefront**, which points to **MyStore.Custom**. The main purpose of **MyStoreFakeXA** is to override the database connection to be compliant with `startSQLRepository`. The module also contains the JAR file with the Oracle connector class and adds it to the class path.

In general, to test repository changes, you will:

1. Open a terminal window and navigate to `/ae101/ATG10.1.1/home/bin`.
2. Issue the following command (all on one line):

   "./startSQLRepository –m MyStoreFakeXA –repository *<full path to repository component>*"

For example, to test the changes to the Profile Repository in this chapter, the command would be:

"./startSQLRepository –m MyStoreFakeXA –repository /atg/userprofiling/ProfileAdapterRepository"

(all in one line)

Guided Instruction

Task: Extending the user profile

1. The `mys_user` table has already been created for you in the `educommprod` schema of the Oracle database (as well as `educommpub`). View this table by using `SQL*Plus`. To do so:
   - Open a terminal window and navigate to `/ae101/oracle-xe/app/oracle/product/11.2.0/xe/bin`.
   - Issue the "`source ./oracle_env.sh`" command to load the Oracle environment variables.
   - Issue the "`./sqlplus`" command to launch the `SQL*Plus` client.
   - Log in with the user/password as either `educommprod/educommprod` or `educommpub/educommpub`.
   - Issue the "`describe mys_user`" command to view details about the `mys_user` table.
   - Take a moment to familiarize yourself with all custom tables for this course. Start with listing out all the table names by issuing the "`select table_name from user_tables where table_name like 'MYS%';`" command.

2. In Eclipse, select the **Package Explorer tab** browser and create a file named `/atg/userprofiling/userProfile.xml` under the **MyStore.Custom**'s `config` folder. You will need to create the `userprofiling` folder. (**Note:** At any point during this course when you are asked to create a file in a certain folder path, create any new folders necessary.)

Tip: If you use Eclipse and create a file of type XML, make sure to erase the XML declaration it adds automatically for files of this type. This declaration may interfere with XML layering combination.

3. Add the following content to this file and save the file:

```
<gsa-template>
 <item-descriptor name="user">

  <table name="mys_user" type="auxiliary"
   id-column-name="user_id">

   <property name="points"
    display-name="Loyalty Points"
    category="My Store Extensions" data-type="int"
    column-name="points" default="0" />

   <property name="mystore"
    display-name="My Home Store"
    category="My Store Extensions" item-type="store"
    repository="/atg/store/stores/StoreRepository"
```

```
        column-name="store_id"/>

      </table>
     </item-descriptor>
    </gsa-template>
```

4. To test your changes:
   - Open a terminal window and navigate to `/ae101/ATG10.1.1/home/bin`.
   - Issue the following command (all on one line):

     ```
     "./startSQLRepository -m MyStoreFakeXA -repository
     /atg/userprofiling/ProfileAdapterRepository"
     ```

     Resolve any errors or warnings before proceeding.
   - Close the terminal window.

5. In Eclipse, select **Project > Build All** to build the **MyStore.Custom** project.
6. Open your **CIM** window and assemble the `atg_production.ear` EAR
7. Open your WebLogic Admin UI browser window and deploy the `atg_production.ear` EAR to WebLogic.

Note: If the production server is down, restart and watch the console log for successful startup messages. If you closed either the CIM terminal window or the WebLogic Admin UI browser window, refer to Practice 2-2, Task 3.

8. In a browser window, enter the URL `http://localhost:7103/dyn/admin` to open the ATG Administration page. There will be two pop-up windows. The first one is for the Dynamo Admin UI login. The Dynamo login is `admin/password1`. The second pop-up is for the Weblogic Server login. The Weblogic login is `weblogic/password1`.
9. Click **ATG Control Center (ACC) Administration** to open the ACC Administration page.
10. Click **Start ACC in Server VM** to start the ACC. The login for the ACC is also `admin/password1`.
11. In the **ACC**, navigate to **People and Organizations > Users** and click the **List** button.
12. Select `lisa@example.com` and check that the **Loyalty Points** and **My Home Store** properties are visible under the **My Store Extensions** heading (you will have to scroll down). Fill in a number of points and select a store (click the ellipsis button in the **My Home Store** property).

    After you have selected a store, double-click its name in the profile to edit the store's properties. Change the country from `USA` to `US` (this is for a later lab).

    Save the profile.

Practices for Lesson 4: Extending Scenarios for Commerce

Chapter 4

Practice 4-1: Customizing Scenarios

Overview

In this practice, you will be creating a custom scenario that will award points to a customer during the registration process. At the end of this exercise you will be able to:

- Create a custom scenario action
- Create a scenario

Problem Statement

You will be creating a custom action that will allow business users to create scenarios that award points to users. When the action is used, the person creating the scenario will be prompted to specify how many points should be awarded. The profile property to which to add the points should be configurable by administrators in case it changes (which it will later in the course).

Planning

Take a moment before looking at the next page to think about what you will need to do to accomplish the task discussed above. Try to list the extensions and customizations you will need to make in as much detail as possible, including pseudo-code. Refer back to the slides if needed. If you want, you may do this part of the exercise in pairs or groups with your classmates as a discussion.

After you are done, check your plan against the completed plan on the next page. The completed plan includes suggested names for tables, classes, and properties.

Task: Creating a custom scenario action

- Create a new class for the configuration component named `AddPointsActionConfig`.
  - Extends `atg.nucleus.GenericService`
  - In the `com.mystore.points` package
  - Has one property, `pointsPropertyName` of type `String`, for storing the name of the points property, with associated setter and getter methods.

- Create a new class for the action named `AddPointsAction`.
  - Extend `atg.process.action.ActionImpl` in the `com.mystore.points` package
  - Import `java.util.Map`, `atg.process.ProcessExecutionContext`, `atg.process.ProcessException`, `atg.beans.DynamicBeans`, `atg.repository.MutableRepositoryItem`, and `atg.repository.MutableRepository`
  - Have one member variable to store the `AddPointsActionConfig` object.
  - In `initialize()`, store a required parameter named `points`.
  - In `configure()`, store the `AddPointsActionConfig` object in member variable.
  - In `executeAction()`:
    - Get number of points to add from `points` scenario parameter stored in `initialize()`
    - Get profile from `ProcessExecutionContext` as a `MutableRepositoryItem`
    - Get Profile Repository as a `MutableRepository` from the profile by using `getRepository()`
    - Retrieve name of points property from `AddPointsActionConfig`
    - Get user's current number of points from their profile by using `DynamicBeans.getSubPropertyValue(profile, pointsProperty)` (this method works for both direct properties and properties of properties); will need to cast object returned as `Integer`
    - Add scenario's points value to user's current points balance
    - Use `DynamicBeans.setSubPropertyValue(profile, pointsProperty, newPoints)` to set new points balance
    - Call `profileRepository.updateItem(profile)` to save changes to the profile
    - Use `AddPointsActionConfig`'s `isLoggingDebug` and `logDebug` methods to output debugging messages as appropriate

- Create a new component named `/com/mystore/AddPointsActionConfig` under the **`MyStore.Custom`** `config` layer.
  - Scope is global.

- It is based on `com.mystore.AddPointsActionConfig`.
- The value of `pointsPropertyName` is `points`.
- The value of `loggingDebug` is `true`.

- Extend `scenarioManager.xml` under the **`MyStore.Custom`** `config` layer to configure the new action.
  - Name of action is `Add Points`.
  - Action class is `com.mystore.AddPointsAction`.
  - Action configuration is `/com/mystore/AddPointsActionConfig`.
  - Execution policy is `individual`.
  - Error response is `continue`.
  - One action parameter named `points`, an Integer that is required. Set the `displayName` as `Number to add`.

- Testing:
  - Rebuild project, assemble `atg_production.ear`, and deploy EAR to WebLogic. Restart server if it is not already running.
  - Create a new scenario that awards 200 points on registration.
  - Wait five minutes for scenario to become active, and then register a new user. Verify in the ACC that new user has 200 points.

You are encouraged to try to complete this exercise by using only the plan above. You should use the suggested variable, class, and component names.

Alternatively, there are step-by-step instructions starting on the next page if you prefer. Whichever method you choose, there are solutions available for reference in the `/ae101/Training/COM/solutions` for each chapter should you need them.

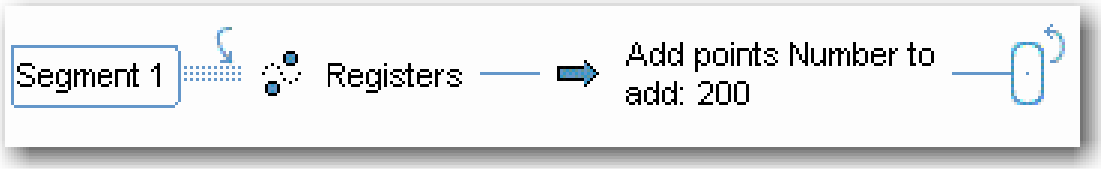
Guided Instruction

Task: Creating a custom scenario action

1. In Eclipse, switch to the **Java EE** perspective and create a class called `AddPointsActionConfig`. This class should be in the `com.mystore.points` package and extend `atg.nucleus.GenericService`.
2. Implement a String property in this class called `pointsPropertyName`, with a get and set method.
3. Save the `AddPointsActionConfig` class. Click **Project > Build All** in the Eclipse menu. Watch the Problems tab below the code window in Eclipse for any compilation errors. To build automatically in Eclipse in the future, you may also check the **Project > Build Automatically** box.
4. Copy the `AddPointsAction.java` file from the `<coursedir>/src/Lesson04` directory to `Mystore/Custom/src/com/mystore/points`. Refresh the view in Eclipse and open this file.
5. There are four comments in the `AddPointsAction` code that start with "`//add code to…`". After each of these comments, add the appropriate code called for in the comment. Refer to the code samples in the slides for help. (**Note:** There are also other comments in this file, which you can ignore.)
6. Save the `AddPointsAction` class. Rebuild and watch the Problems tab for errors.
7. Switch to the **ATG** perspective and go to the **ATG Component Browser** tab. Right-click **`MyStore.Custom`** and select **New Component**.
8. Type `com.mystore.points.AddPointsActionConfig` in the **Class name** field, or browse to the class. Type `/com/mystore/points/AddPointsActionConfig` in the **Component name** field.
9. Check **Edit Component after wizard completion** and click **Finish**.
10. Set the value of `pointsPropertyName` to `points`. Set the value of `loggingDebug` to `true`. Click **Save**.
11. Open a terminal window and navigate to `/ae101/ATG10.1.1/MyStore/Custom/config/atg/`. If it does not already exist, create a subdirectory `scenario` by typing `"mkdir scenario"`.
12. In the same window, navigate to `/ae101/Training/COM/src/Lesson04`.
13. Copy the `scenarioManager.xml` file to **`MyStore.Custom`** by typing `"cp scenarioManager.xml /ae101/ATG10.1.1/MyStore/Custom/config/atg/scenario"`. (At any point during this course when you are asked to create a file in a certain folder path, create any new folders necessary.) This file configures the new action. If you want, open and review the contents of the file.
14. In Eclipse, rebuild the project, assemble `atg_production.ear` and deploy EAR to WebLogic. Restart server if it is not already running.

Note: If you make changes to the project outside of Eclipse such as creating new files on the file system directly, in Eclipse, right-click the project name and select `refresh` or press the **F5** key to refresh the project before building.

15. Start the **ACC** and navigate to **Scenarios > Scenarios**. Expand the `store` folder and you will find a folder named `points`.
16. Select the `points` folder and click the **New Scenario** button. Select **No** to using a scenario template and click **Next**. Name the new scenario `PointsOnRegistration` and click **Finish**.
17. Create this scenario:



Tip: Move your mouse pointer over the line between the segment label and the end of the scenario. Click the diamond that appears on the line. This will bring up a menu of scenario building blocks. If you need more assistance, ask your instructor for help.

18. Save and enable the scenario by going to **File > Enable Scenario**.
19. Wait five minutes for the Scenario Manager to load the new scenario. If you want, navigate to `/atg/scenario/ScenarioManager` in the admin browser's **Component Browser** section to verify that `PointsOnRegistration` has been added to the list of running scenarios. If not, make sure that it is saved and enabled, then refresh the Scenario Manager page in another few minutes.
20. Go to the store by using the **Visit Production Store** shortcut and register as a new user (**Tip:** Click the **Login** link in the upper right corner and use the **New Customer** section to begin registration).
21. Use the **ACC** to look up your new user. You should see that the user has 200 points. If not, ask your instructor how to enable debugging for scenarios and then test registering another new user. You can also use the debugging messages printed on the server console by `AddPointsActionConfig` to troubleshoot the issue.

Practices for Lesson 5: Displaying and Extending the Catalog

Chapter 5

Practice 5-1: Extending the Product Catalog

Overview

In this practice, you will be extending the product catalog. At the end of this exercise you will be able to:

- Add custom properties to the product catalog
- Update a product property
- Create and deploy a project in the **BCC**

Problem Statement

The store needs to integrate with a tax calculator to determine the amount of tax on an order. Some items in the catalog will be exempt from tax. This will be identified as a Boolean property on a product.

Planning

Take a moment before looking at the next page to think about what you will need to do to accomplish the tasks discussed above. Try to list the extensions and customizations you will need to make in as much detail as possible, including pseudo-code. Refer back to the slides if needed.

After you are done, check your plan against the completed plan on the next page. The completed plan includes suggested names for tables, classes, and properties.

Plan

Task 1: Extend the `product` repository item.

- Create a new table named `mys_product` in the production and publishing data sources for custom properties to be added to profile.
  - The new table will have two columns: `product_id` (`varchar2(40)`) and `taxfree` (`number(1)`).
  - The new table in the publishing schema will have the additional column for versioning: `asset_version(number(19))`.
  - **Note:** Remember that all database tables have already been created for you. Step 1 in the guided instructions section describes how to view the tables.
- Extend `productCatalog.xml` under the **`MyStore.Custom`** `config` layer to model new properties.
  - Item descriptor to extend is `product`.
  - Table is auxiliary type.
  - `taxfree` is a Boolean property with a default value of `false`.

- Testing:
  - Use `startSQLRepository` to test Product Catalog extension.
  - Build your project in Eclipse, assemble and install the new `atg_production` and `atg_publishing_lockserver` EAR files by using CIM, and deploy both EAR files by using the WebLogic Admin Server.
  - If they are not already running, start the production and publishing servers and watch for errors.

Task 2: Update the Men's Leather Jacket tax-free property.

- Use the **BCC** to update a product's tax-free property.
  - Create a project and edit the Men's leather jacket and set it to tax free.
  - Deploy project to production.

You are encouraged to try to complete this exercise by using only the plan above (you will not need to create the database tables, as they have already been created for you. Use `SQL*Plus` and log in to the database schemas for both production and publishing schemas to view the database table `mys_product`). You should use the suggested variable, class, and component names.

Alternatively, there are step-by-step instructions starting on the next page if you prefer. Whichever method you choose, there are solutions available for reference in the `Training` folder should you need them.

Guided Instruction

Task 1: Extend the `product` repository item.

1. The `mys_product` table has already been created for you in the `educommprod` and `educommpub` (with version) schemas of the Oracle database. View this table by using `SQL*Plus`. To do so:
   - Open a terminal window and navigate to `/ae101/oracle-xe/app/oracle/product/11.2.0/xe/bin`.
   - Issue the "`source ./oracle_env.sh`" command to load the Oracle environment variables.
   - Issue the "`./sqlplus`" command to launch the `SQL*Plus` client.
   - Log in using the user/password as either `educommprod`/`educommprod` or `educommpub`/`educommpub`.
   - Issue the "`describe mys_product`" command to view details about the `mys_product` table.
   - The educommpub schema includes a column to store the version number.

2. In Eclipse, create a file named `/atg/commerce/catalog/custom/customCatalog.xml` under the **`MyStore.Custom`** `config` folder. You will need to create the `commerce/catalog/custom` folder (**Tip:** You can create all three folders in one step).

Tip: If you use Eclipse and create a file of type XML, make sure to erase the XML declaration it adds automatically for files of this type. This declaration may interfere with XML layering combination.

3. Add the following content to this file:

```
<gsa-template>
  <item-descriptor name="product" xml-combine="append">
    <table name="mys_product" type="auxiliary"
     id-column-name="product_id">
      <property name="taxfree" display-name ="Tax Free Flag"
         category="My Store Extensions"
         data-type="boolean"
         column-name="taxfree"
         default="false" />
    </table>
  </item-descriptor>
</gsa-template>
```

4. To test your changes:
   - Open a terminal window and navigate to `/ae101/ATG10.1.1/home/bin`.
   - Issue the following command (all on one line):

     ```
     ./startSQLRepository -m MyStoreFakeXA -repository
     /atg/commerce/catalog/ProductCatalog
     ```

     Resolve any errors or warnings before proceeding.
   - Close the terminal window.

5. Rebuild project, assemble both `atg_production.ear` and `atg_publishing_lockserver.ear` files, and deploy to WebLogic. Restart both servers if they are not already running.

   Note: The catalog repository is a versioned repository which means all repository configurations require both servers to be rebuilt. This allows catalog updates to be made in the **BCC** and published to the production server.

Task 2: Update the Men's Leather Jacket tax-free property.

.

6. In **Comm_Shortcuts**, double-click the **Visit BCC** shortcut. This shortcut opens the ATG Business Control Center (BCC) login screen in the browser. The login for the BCC is `admin/password1`.
7. Expand the **Merchandising** section of the **Operations** menu and click **Manage Commerce Assets**.
8. Enter `Update tax-free product` as the project name and click **Continue**.
9. Click the arrow next to **Site Catalogs** to view the catalogs in the product catalog repository. Navigate down the tree **ATG Store > Store Catalog Folder > Master Catalog > Commerce Root > Men > Jackets** to the list of Men's jackets.
10. Double-click **Leather Jacket** on the left to view its properties. Click the **Advanced** tab in the **Edit Product** panel. Note the **Tax Free Flag** field, set to `No`.
11. Click the **Tax Free Flag** property, set it to `Yes`, and click **Save**.
12. In the top middle of the screen, click the drop-down next to **Author** and select **Ready for Review**. In the pop-up confirmation box, click **Continue**.
13. The project stage has now changed to **Content Review**. Click the drop-down next to the stage name and select **Approve Content**. Again, click **Continue** in the pop-up confirmation box.
14. The project stage has now changed to **Approve for Production Deployment**. Click the drop-down next to the stage name and select **Approve and Deploy to Production**. Again, click **Continue** in the pop-up confirmation box. Click the **Home** button in the following pop-up.
15. If you want, switch to the CA server console to view the progress of the deployment. When the deployment is finished, you will see a message similar to the following

    ```
    **** info       Tue Dec 11 14:02:36 EST 2012    1355252556426
    /atg/deployment/DeploymentManager       Purging deployment data
    for deployment 1400002
    ```

```
**** info       Tue Dec 11 14:02:36 EST 2012    1355252556653
/atg/deployment/DeploymentManager       Deployment 1400002
finished in 0:13.460
```

Practices for Lesson 6 Customizing and Extending Multisites

Chapter 6

Practice 6-1: Extending the Site Repository

Overview

Practices titled “Customizing and Extending Multisite” and “Creating and Using New Site Categories” cover the multisite capability. In this practice, you will be extending the site repository and viewing these properties in the **BCC**. In the next practice, you will apply these customizations to create a reward site. At the end of this exercise you will be able to:

- Extend the `siteConfiguration` repository item
- Extend the `siteTemplate` item
- View repository updates in the **Dynamo Administration UI** page
- Hide fields or make them read-only in the **BCC**

Problem Statement

MyStore will be using the multisite feature to implement its reward sites.

Orders on reward sites will not be eligible to earn points or be discounted by any promotions. In order to implement these features, the application will need to be able to distinguish between normal sites and reward sites through a `boolean` property.

BCC users should see two site templates or categories: one for normal sites and one for reward sites. The sites started with the reward category should have the reward property set to `true` by default; for normal sites, it should be set to `false`. The display of this `boolean` property should be read-only in the **BCC**. In addition, because **MyStore** is not running ATG Self-Service, the `siteTypes` property should be hidden for all sites. Any new text displayed in the **BCC** should be internationalizable.

Planning

Take a moment before looking at the next page to think about what you will need to do to accomplish the tasks discussed above, as you have in the previous labs. A completed plan begins on the next page.

Plan

Task 1: Extend the `siteConfiguration` repository item.

- Create new table called `mys_site` in production and publishing data sources for custom properties to be added to site.
  - The new table will have two columns: `site_id` (`varchar2(40)`) and `is_reward` (`number(1)`).
  - The new table in the publishing schema will have the additional column for versioning: `asset_version(number(19))`.

- Create new resource bundle under the **`MyStore.Custom`** `src` for new **BCC** labels.
  - Resource bundle will be named `com.mystore.multisite.SiteRepositoryTemplateResources`.
  - Add the text to be used for label of the reward field in BCC; key is `reward`.

- Extend `siteRepository.xml` under the **`MyStore.Custom`** `config` layer to model new property.
  - The item descriptor to extend is `siteConfiguration`.
  - Table is auxiliary type.
  - `reward` is a `boolean` property.
  - Resource bundle key is `reward`.

- Testing:
  - Use `startSQLRepository` to test extension to `siteConfiguration`.

Task 2: Extend the `siteTemplate` repository item.

- Create a new table called `mys_site_template` in production and publishing datasources for custom properties to be added to site.
  - The new table will have two columns: `site_template_id` (`varchar2(40)`) and `default_reward_value` (`number(1)`).
- Add text to `com.mystore.multisite.SiteRepositoryTemplateResources` for label of field for default reward value in site category configuration.
  - The key is `default_reward`.

- Edit `siteRepository.xml` under the **`MyStore.Custom`** `config` layer to model new property.
  - The item descriptor to extend is `siteTemplate`.
  - Table is auxiliary type.
  - `reward` is a `boolean` property.
  - The resource bundle key is `default_reward`.

- Testing:
  - Use `startSQLRepository` to test extension to `siteTemplate`.

Task 3: Alter the display of `siteConfiguration` properties in the BCC.

- Alter `siteTypes` property to be hidden.
  - Table is `site_types`.
  - Set `hidden` attribute to `true`.

- Alter `reward` property to be read-only.
  - Add `uiwritable` attribute set to `false`.

- Testing:
  - Use `startSQLRepository` to test extension to `siteConfiguration`.
  - Build, assemble, and deploy EAR files to the production and publishing servers.
  - View repository properties in the **BCC**.

You are encouraged to try to complete this exercise by using only the plan above (you will not need to create the database tables, as they have already been created for you). You should use the suggested variable, class, and component names.

Alternatively, there are step-by-step instructions starting on the next page if you prefer.

Guided Instruction

Task 1: Extend the `siteConfiguration` repository item.

1. The `mys_site` table has already been created for you in the `educommprod` and `educommpub` schemas of the Oracle database. View this table by using `SQL*Plus`. To do so:
   - Open a terminal window and navigate to `/ae101/oracle-xe/app/oracle/product/11.2.0/xe/bin`.
   - Issue the "`source ./oracle_env.sh`" command to load the Oracle environment variables.
   - Issue the "`./sqlplus`" command to launch the `SQL*Plus` client.
   - Log in using the user/password as either `educommprod/educommprod` or `educommpub/educommpub`.
   - Issue the "`describe mys_site`" command to view details about the `mys_site` table.
2. Using a text editor or Eclipse, create a file named `/atg/multisite/siteRepository.xml` under the **`MyStore.Custom`** `config` folder.
3. Add the following content to this file and save it:

```
<gsa-template>
  <item-descriptor name="siteConfiguration">
    <table type="auxiliary" name="mys_site"
      id-column-name="site_id">

      <property name="reward" data-type="boolean"
        column-name="is_reward"
        category="My Store Extensions"
        display-name-resource="reward" default="false">

        <attribute name="resourceBundle"
          value="com.mystore.multisite.
          SiteRepositoryTemplateResources"/>
```

(**Note:** In the preceding tag, the `value` attribute and its value should be on one line.)

```
      </property>
    </table>
  </item-descriptor>
</gsa-template>
```

4. Using a text editor or Eclipse, create a file named `/com/mystore/multisite/SiteRepositoryTemplateResources.properties` under the **`MyStore.Custom`** `src` folder.

Note: Although this is a properties file, it is **not** an ATG component. It is a Java resource bundle, and therefore it needs to go in the `src` folder, **not** the `config` folder.

5. Add the following content to this file and save it:

```
reward=Is this a reward site?
```

6. To test your changes:
   - Open a terminal window and navigate to `<atgdir>/home/bin`.
   - Issue the following command (all on one line):

     ```
     startSQLRepository -m MyStoreFakeXA -repository
     /atg/multisite/SiteRepository
     ```

     Resolve any errors or warnings before proceeding.
   - Leave the window open for later use.

7. Rebuild project, assemble both `atg_production.ear` and `atg_publishing_lockserver.ear` files, and deploy to WebLogic. Restart both servers if they are not already running. **Note:** The site repository is also versioned like the catalog so we need to rebuild both servers.
8. After the server has finished starting, in a browser window, enter the URL http://localhost:7103/dyn/admin to open the ATG Administration page. The ATG login is `admin/password1`. The WebLogic login is `weblogic/password1`.
9. Click the **Component Browser** link, and then browse down to `/atg/multisite/SiteRepository`.
10. Next to the `siteConfiguration` item, click **See Property Descriptions**. Scroll down and verify that the `reward` property is displayed. The name of the property should be displayed as "Is this a reward site? (reward)". The default value should be `false`.

    Tip: Leave this page open for testing later changes.

Task 2: Extend the `siteTemplate` repository item.

11. The `mys_site_template` table has already been created for you in the `educommprod` and `educommpub` schemas of the Oracle database. View this table by using `SQL*Plus`.
12. Using a text editor or Eclipse, edit `/atg/multisite/siteRepository.xml` under the **`MyStore.Custom`** `config` folder.
13. Add the following content below the closing `</item-descriptor>` tag for `siteConfiguration` that you added earlier and above `</gsa-template>` and save file:

```
<item-descriptor name="siteTemplate">
  <table type="auxiliary" name="mys_site_template"
    id-column-name="site_template_id">

    <property name="reward" data-type="boolean"
```

```
        column-name="default_reward_value"
        category="My Store Extensions"
        display-name-resource="default_reward"
        default="false">

      <attribute name="resourceBundle"
        value="com.mystore.multisite.
        SiteRepositoryTemplateResources"/>
```

(**Note:** In the preceding tag, the `value` attribute and its value should be on one line)

```
      </property>
    </table>
  </item-descriptor>
```

14. Using a text editor or Eclipse, edit `/com/mystore/multisite/SiteRepositoryTemplateResources.properties` under the **`MyStore.Custom`** `src` folder.

    Add the following content to this file and save it:

    ```
    default_reward=Default value of reward property
    ```

15. To test your changes:
    - Open the terminal window used in step 6.
    - Issue the following command (all on one line):

      ```
      startSQLRepository -m MyStoreFakeXA -repository
      /atg/multisite/SiteRepository
      ```

      Resolve any errors or warnings before proceeding.
    - Close the terminal window.
16. Rebuild the project, assemble both `atg_production.ear` and `atg_publishing_lockserver.ear` files, and deploy to WebLogic. Restart both servers if they are not already running.
17. After the server has finished starting, in a browser window, enter the URL http://localhost:7103/dyn/admin to open the ATG Administration page. The login is `admin/password1`. Alternatively, if you left this page open from earlier in the exercise, scroll back to the top of the page and refresh it.
18. Click the **Component Browser** link, and then browse down to `/atg/multisite/SiteRepository`.
19. Next to the `siteTemplate` item, click **See Property Descriptions**. Scroll down and verify that the `reward` property is displayed. The name of the property should be displayed as "Default value of reward property (reward)". The default value should be `false`.

Task 3: Altering the display of `siteConfiguration` properties in the BCC

20. To view the current display of sites in the BCC, open the **BCC** by using the shortcut provided (**Visit BCC**). The login is `admin/password1`.

21. Expand the **Site Administration** section of the **Operations** menu and click **Manage Site Assets**.
22. Enter `Create US Reward Site` as the project name (we will come back and use this project later to add the new US reward site) and click **Continue**.
23. Double-click one of the existing sites on the left to open its properties on the right.
24. Scroll down the properties. Note the display of the **Site Types** property. Scroll to view the `reward` property. If you click it, it becomes editable. Log out of the **BCC**.
25. Using a text editor or Eclipse, edit `/atg/multisite/siteRepository.xml` under the **`MyStore.Custom`** `config` folder.
26. Add the text in bold below to the existing XML:

```
<item-descriptor name="siteConfiguration">
  <table type="auxiliary" name="mys_site"
    id-column-name="site_id">

    <property name="reward" data-type="boolean"
      column-name="is_reward"
      category="My Store Extensions"
      display-name-resource="reward" default="false">

      <attribute name="uiwritable" value="false"/>

      <attribute name="resourceBundle"
          value="com.mystore.multisite.
          SiteRepositoryTemplateResources"/>
       </property>
    </table>

    <table name="site_types">
      <property name="siteTypes" hidden="true"/>
    </table>
  </item-descriptor>
```

27. To test your changes:
    - Open a terminal window and navigate to `<atgdir>/home/bin` (or reuse the one opened earlier to test the previous changes).
    - Issue the following command (all on one line):

      ```
      startSQLRepository -m MyStoreFakeXA -repository
      /atg/multisite/SiteRepository
      ```

      Resolve any errors or warnings before proceeding.
    - Close the terminal window.

28. Rebuild project, assemble both `atg_production.ear` and `atg_publishing_lockserver.ear` files, and deploy to WebLogic. Restart both servers if they are not already running.
29. Log in to the **BCC** again.
30. Select **All Projects and Tasks** in the **To Do List** section. Expand the **Create US Reward Site** project and click the **Author** link.
31. Double-click a site to view its properties. Note that the **Site Types** property does not appear, and the reward property is read-only.

Practices for Lesson 7 Using Multisites to Create a New Site

Chapter 7

Practice 7-1: Creating a New Site

Overview

In the practice titled “Customizing and Extending Multisite,” you extended the site repository with additional properties required for a reward site. In this practice, you will be creating a new site in the BCC. At the end of this exercise, you will be able to:

- Create an abstract shareable type
- Create a new site category
- Create new site groups

Problem Statement

Reward sites will not share a cart with normal sites, and will have their own price lists. However, reward sites and their alternate normal site (such as **ATG Store** and **US Reward Site**) should be grouped together for cross-linking on the site pages.

Planning

Take a moment before looking at the next page to think about what you will need to do to accomplish the tasks discussed above, as you have in the previous labs. A completed plan begins on the next page.

Plan

Task 1: Create a new site category.

- Create a new site category for the reward sites. It should use the same Item Mapping Name as the Default Category.
  - Set the default value of `reward` to `true`.

- Testing:
  - Create a reward site for the US.
    - ID is `RewardUS`.
    - Name is `US Reward Site`.
    - Site priority is `2`.
    - Site base URL is `/crs/rewardus`.
    - Price list is new price list named `Reward US`.
    - **Note:** All other values will be the same as in the **ATG Store site**.

Task 2: Create an abstract shareable type.

- Create a component based on `atg.multisite.ShareableType`.
  - The name of the component is `/com/mystore/multisite/RelatedRewardSite`.
  - displayName is `Related Reward Site`.
  - id is `com.mystore.reward`.

- Add `/com/mystore/multisite/RelatedRewardSite` to `/atg/multisite/SiteGroupManager.shareableTypes`.

Task 3: Create new site groups.

- Create a Reward Sites site group.
  - US Reward Site is only member.
  - Site group shares the cart.

- Create a Related US Sites group.
  - US Reward Site and ATG Store are the members.
  - Site group shares `Related Reward Site`.

Task 4: Create a link to the Reward Sites on Main Store page.

- Create a component based on `atg.droplet.multisite.SharingSitesDroplet`.
  - The name of the component is `/com/mystore/multisite/RewardSharingSitesDroplet`.
  - Set the `siteGroupManager` and `siteContextManager` properties to the ATG components.
  - Set the `shareableTypeId` property to `com.mystore.reward`.

- Edit `/navigation/gadgets/sites.jsp` to include links between regular site and reward site.
  - Import the `RewardSharingSitesDroplet`.
  - Copy existing `CartSharingSitesDroplet` and its contents.
  - Change the name of `CartSharingSitesDroplet` to `RewardSharingSitesDroplet`.

You are encouraged to try to complete this exercise by using only the plan above. You should use the suggested variable, class, and component names.

Alternatively, there are step-by-step instructions starting on the next page if you prefer.

Guided Instruction

Task 1: Create a new site category.

1. Open the **BCC**.
2. Expand the **Site Administration** section of the **Operations** menu and click **Manage Site Categories**.
3. Double-click the **Default Category** on the left to view its properties. Note the **Default value of reward property** field, set to `No`.
4. Click the new item button in the upper left corner ( ) and select **Site Category**.
5. Enter the following values:

   Category Name: `Reward`
   Category Description: `For reward sites`
   Item Mapping Name: `SiteCategory:default`
   Default value of reward property: `Yes`
6. Click the **Create** button.
7. Click the **Site Category Administration** drop-down in the upper left corner and select **Home**.
8. Under **To Do List**, select **All Projects and Tasks**. Expand the **Create US Reward Site** project and click the **Author** link. This will return you to the Site Administration screen.
9. Click the arrow in the lower left corner ( ) to open the split-screen view. In the lower screen, double-click to open the **ATG Store** site.
10. In the upper screen, click the new item button and select **Site**. Select **Reward** as the new site's category and click **OK**.
11. Enter the following values for the new site:

    ID: `RewardUS`
    Site Name: `US Reward Site`
    Site Priority: `2`
    Site Base URL: `/crs/rewardus`

12. Click the **Operations** tab.
13. Scroll down to the **Default Catalog** field and click to make it editable. Click the drop-down arrow next to **Select...** and select **Select**.
14. In the pop-up window, click **Find** to list the available catalogs. Highlight the **Master Catalog** option and click **OK**.
15. Scroll down to the **Default List Price List** field and click to make it editable. Click the drop-down arrow next to **Select...** and select **Create Price List**.
16. In the pop-up window, enter `Reward US` in the **Name** field and click **Create**.
17. Fill in the rest of the values for the new site as they are in the ATG Store site, open in the lower screen for reference. You can copy and paste the values of text fields from the lower to the upper screen.
18. Click the **Create** button to save the new US Reward site. Close the split-screen. Log out of the BCC.

Task 2: Create an abstract shareable type.

19. In Eclipse, switch to the ATG perspective and the ATG Component Browser tab.
20. Expand **/com/mystore** and right-click the folder. Select **New Component**.
21. In the **Class name** field, enter or browse to `atg.multisite.ShareableType`.
22. In the **Component name** field, enter `/com/mystore/multisite/RelatedRewardSite`.
23. Check **Edit component after wizard completion** and click **Finish**.
24. In the component editor, enter the following values:

    displayName: `Related Reward Site`
    id: `com.mystore.reward`
25. Click **Save**.
26. Expand **/atg/multisite** and locate the **SiteGroupManager** component. Right-click it and select **Edit Component**.
27. Right-click the **shareableTypes** property and select the **Edit** option, highlight `/atg/commerce/ShoppingCartShareableType`, and select **Insert After**. At the beginning of the line, type `/com/mystore/multisite/RelatedRewardSite` followed by **OK** and **Save**.
28. Rebuild project, assemble both `atg_production.ear` and `atg_publishing_lockserver.ear` files, and deploy to WebLogic.

Task 3: Create new site groups.

29. Log in to the **BCC** again.
30. Select **All Projects and Tasks** in the **To Do List** section. Expand the **Create US Reward Site** project and click the **Author** link.
31. Click the new item button and select **Site Group**. Enter `Reward Sites` in the **Site Group Name** field. Add **US Reward Site** to the **Sites in Group** field by using drag and drop. Check **Shopping Cart** in the **Shared Data** field. Click the **Create** button.
32. Click the new item button and select **Site Group** again. Enter `Related US Sites` in the **Site Group Name** field. Add **US Reward Site** and **ATG Store** to the **Sites in Group** field. Check **Related Reward Site** in the **Shared Data** field. Click the **Create** button.
33. In the top middle of the screen, click the drop-down next to **Author** and select **Ready for Review**. In the pop-up confirmation box, click **Continue**.
34. The project stage has now changed to **Content Review**. Click the drop-down next to the stage name and select **Approve Content**. Again, click **Continue** in the pop-up confirmation box.
35. The project stage has now changed to **Approve for Production Deployment**. Click the drop-down next to the stage name and select **Approve and Deploy to Production**. Again, click **Continue** in the pop-up confirmation box. Click the **Home** button in the following pop-up.
36. If you want, switch to the publishing server console to view the progress of the deployment. When the deployment is finished, you will see a message similar to:

```
**** info       Wed Dec 12 12:34:09 EST 2012    1355333649394
/atg/deployment/DeploymentManager       Purging deployment data
for deployment 1600002
**** debug      Wed Dec 12 12:34:09 EST 2012    1355333649421
/atg/epub/DeploymentServer      Run second apply phase: false
**** info       Wed Dec 12 12:34:09 EST 2012    1355333649473
/atg/deployment/DeploymentManager       Deployment 1600002
finished in 0:13.294
```

37. To test your new site, close all browser windows, open a new browser window, and go to the following URL: `http://localhost:7103/crs/rewardus`. Bookmark the reward site for future use.
38. The store should appear without the home page splash image, because we did not edit the appropriate targeter to create a rule set for this site. Also, the prices of the items on the home page will be blank because we have not yet filled in the site's price list.
39. In the **BCC**, under **To Do List**, expand the **Create US Reward Site** project. Click the **Verify Production Deployment** link.
40. In the Site Administration screen, click the drop-down next to the stage name (**Verify Product...**) and select **Accept Production Deployment**. Again, click **Continue** in the pop-up confirmation box. Click the **Home** button in the following pop-up.

Task 4: Create a link to the Reward Sites on Main Store page.

41. In Eclipse, switch to the ATG perspective and the ATG Component Browser tab, if you are not already there.
42. Expand **/com/mystore/multisite** and right-click the folder. Select **New Component**.
43. In the **Class name** field, enter or browse to `atg.droplet.multisite.SharingSitesDroplet`.
44. In the **Component name** field, enter `/com/mystore/multisite/RewardSharingSitesDroplet`.
45. Check **Edit component after wizard completion** and click **Finish**.
46. In the component editor, enter the following values and save component:

    shareableTypeId: `com.mystore.reward`
    siteContextManager: `/atg/multisite/SiteContextManager`
    siteGroupManager: `/atg/multisite/SiteGroupManager`

47. Open a terminal window and navigate to the `store.war` directory by typing:

    ```
    "cd /ae101/ATG10.1.1/MyStore/Storefront/j2ee-
    apps/Storefront/store.war"
    ```

48. Save a copy of `navigation/gadgets/sites.jsp` to `navigation/gadgets/sites.jsp.ORIG` by typing:

    ```
    "cp navigation/gadgets/sites.jsp
    navigation/gadgets/sites.jsp.ORIG"
    ```

49. In a text editor (for example, vi), open `navigation/gadgets/sites.jsp`.

50. Import the `RewardSharingSitesDroplet` bean.
51. Select the entire `CartSharingSitesDroplet` (starts with `<dsp:droplet name="CartSharingSitesDroplet">` and ends with `</dsp:droplet>` just above `</dsp:page>`). Copy this text.
52. Make a new line just above `</dsp:page>` and paste the copied text. In the new copy, change the `CartSharingSitesDroplet` name to `RewardSharingSitesDroplet`.
53. Rebuild project, assemble the `atg_production.ear` file, and deploy to WebLogic. Restart production server if it is not already running.
54. Load the Store home page. You should now see two sets of links in the upper right corner: one for ATG Store and ATG Home and another for US Reward Site and ATG Store. Click the link for **US Reward Site**. You should now see only a link back to ATG Store.

Practices for Lesson 8: Orders

Chapter 8

Practices for Lesson 8

Practices Overview

There are no practices for this lesson.

Practices for Lesson 9: Extending Orders

Chapter 9

Practice 9-1: Catching Up

Overview

You must complete the practices from the previous lessons in order to do the practices for the rest of this course. If you were unable to complete the previous practices, complete this practice. **If you finished the practices for the previous lessons, skip this practice and go on to Practice 9-2.**

Task 1: Copy the catchup module from the Lesson 9 folder.

1. Close all browsers and bring down all servers (ATG production, publishing, and WebLogic) and CIM terminal windows if they are currently open.
2. In the Eclipse package explorer, select the **`MyStore.Custom`** project and navigate to **Edit > Delete**. Make sure "Delete project contents on disk" is not selected and click **OK**.
3. Close Eclipse.
4. Open a terminal window and navigate to the ATG root directory by typing:
   `"cd /ae101/ATG10.1.1"`
5. Back up the existing **MyStore** project by typing:
   `"mv MyStore MyStore.BAK"`
6. Copy the **MyStore** module from the solutions folder into ATG by typing (on a single line):
   `"cp -R /ae101/Training/COM/solutions/Lesson09/catchup/MyStore ."`

Task 2: Back up the existing database.

7. Make a backup folder in your home directory to store the database files by typing the following commands:
   `"cd ~"`
   `"mkdir dbfiles"`
8. Change the directory to the Oracle executable directory by typing:
   `"cd /ae101/oracle-xe/app/oracle/product/11.2.0/xe/bin"`
9. Set up the Oracle environment variables by typing:
   `"source ./oracle_env.sh"`
10. Back up the production database schema by typing (on a single line):
    `"exp educommprod/educommprod@xe full=y file=~/dbfiles/educommprod.dmp log=~/dbfiles/educommprod.log"`
11. Backup the publishing database schema by typing (on a single line):
    `"exp educommpub/educommpub@xe full=y file=~/dbfiles/educommpub.dmp log=~/dbfiles/educommpub.log"`

Task 3: Import data extracts into the databases.

12. Issue the `"./sqlplus"` command to launch the `SQL*Plus` client and sign in as `educommprod/educommprod`.

13. Drop the `educommprod` schema by typing (on a single line):

```
"@/ae101/Training/COM/solutions/Lesson09/catchup/dbfiles/delete_
my_schema.sql"
```

14. Issue the `"./sqlplus"` command to launch the `SQL*Plus` client and sign in as `educommpub/educommpub`.
15. Drop the `educommpub` schema by typing (on a single line):

```
"@/ae101/Training/COM/solutions/Lesson09/catchup/dbfiles/delete_
my_schema.sql"
```

16. Import the production database schema by typing (on a single line):

```
"imp educommprod/educommprod@xe full=y
file=/ae101/Training/COM/solutions/Lesson09/catchup/dbfiles/educ
ommprod.dmp"
```

17. Import the publishing database schema by typing (on a single line):

```
"imp educommpub/educommpub@xe full=y
file=/ae101/Training/COM/solutions/Lesson09/catchup/dbfiles/educ
ommpub.dmp"
```

Task 4: Configure environment and test.

18. Return to the practice titled "Commerce Reference Store" and follow steps to restart servers, import **`MyStore.Custom`** into Eclipse, assemble and deploy application to WebLogic, and create a link to the store JSP directory.
19. Open a terminal window and navigate to the `store.war` directory by typing:

```
"cd /ae101/ATG10.1.1/MyStore/Storefront/j2ee-
apps/Storefront/store.war"
```

20. Save a copy of `navigation/gadgets/sites.jsp` to `navigation/gadgets/sites.jsp.ORIG` by typing:

```
"cp navigation/gadgets/sites.jsp
navigation/gadgets/sites.jsp.ORIG"
```

21. Copy `sites.jsp` from the solution folder into the `navigation/gadgets` folder by typing:

```
"cp /ae101/Training/COM/solutions/Lesson09/catchup/sites.jsp
navigation/gadgets/sites.jsp"
```

22. Rebuild project, assemble the `atg_production.ear` file, and deploy to WebLogic. Restart production server if it is not already running.
23. Take a moment to familiarize yourself with the solutions for the previous lessons. Run through some of the tests at the end of each lesson to verify that the customizations work as expected and understand the differences between this solution and yours.
24. To test that you are ready to move on to the next practice, close all browser windows, open a new browser window, and go to the following URL: `http://localhost:7103/crs/rewardus`. You should see the **US Reward Site** (without the homepage banner) and links in the header to switch to the **ATG Store.**

Practice 9-2: Customizing Orders and Order Management

Overview

In this practice, you will be customizing orders and order persistence. At the end of this exercise you will be able to:

- Extend the Order class
- Extend the order item descriptor
- Extend order persistence
- Extend the creation of orders
- Set a global component's property based on a site property

Problem Statement

The store needs to distinguish which orders are reward orders for functionality we will add in later labs. To implement this, we will add a boolean `reward` property to the order and extend order persistence to include this property. We will then add the necessary logic to set the `reward` property to the `OrderManager` method that creates orders.

Planning

Take a moment before looking at the next page to think about what you will need to do to accomplish the tasks discussed above, as you have done in previous exercises.

After you are done, check your plan against the completed plan on the next page. The completed plan includes suggested names for tables, classes, and properties.

Plan

Task 1: Extend the order repository and order persistence.

- Create new table called `mys_order` in production and publishing data sources.
  - The new table will have two columns: `order_id` (`varchar2(40)`) and `reward` (`number(1)`).

- Extend `orderrepository.xml` under the **`MyStore.Custom`** `config` layer.
  - Item descriptor to extend is `order`.
  - Table is auxiliary.
  - `reward` is a `boolean` property.
  - Default value of `reward` is `false`.

- Alter the `/atg/commerce/order/processor/LoadOrderObject` component.
  - Add `reward` to its `loadProperties` property.

- Testing:
  - Use `startSQLRepository` to test extension to `order`.
  - Build, assemble, and deploy EAR files to the production and publishing servers.
  - View the `OrderRepository` component through the **Dynamo Admin UI** Component Browser to verify that the `reward` property is live.
  - View the `LoadOrderObject` component and verify that `loadProperties` is correct.

Task2: Extend the order class.

- Create a new class called `com.mystore.order.MyStoreOrder`.
  - Extend `atg.projects.store.order.StoreOrderImpl`. Add get and set methods for a `boolean` reward property.
    - Methods should use `getPropertyValue()` and `setPropertyValue()` to flow through to the `order` repository item.

- Alter the properties of `OrderTools` so that new order class is used.
  - Remove `atg.projects.store.order.StoreOrderImpl=order` from the `beanNameToItemDescriptorMap` property.
  - Add `com.mystore.order.MyStoreOrder=order` to the `beanNameToItemDescriptorMap` property.
  - Add `com.mystore.order.MyStoreOrder` twice to the `orderTypeClassMap` property, with keys `shoppingcart` and `default`.

- Testing:
  - Build, assemble, and deploy EAR files to the production and publishing servers.
  - Verify that `OrderTools` properties are set correctly.
  - Open store in a different browser.
  - Use `/atg/dynamo/servlet/sessiontracking/GenericSessionManager` to view the session-scoped components for the store session. Navigate down to `/atg/commerce/ShoppingCart.current`. Verify that the class of the order object is `com.mystore.order.MyStoreOrder` and the `reward` property is set to `false`.

Task 3: Extend the order manager.

- Create new class called `com.mystore.order.MyStoreOrderManager`.
  - Extend `atg.projects.store.order.StoreOrderManager`.
  - Add get and set methods for a boolean reward property.
  - Override the `createOrder` method of `StoreOrderManager`.
    - Add code to this method to set the new order's reward property based on the value of the Order Manager's reward property (which is set from the site reward property).

- Alter the `/atg/commerce/order/OrderManager` component.
  - Change class to `com.mystore.order.MyStoreOrderManager`.
  - Add necessary lines to properties file to have value of Order Manager's reward property set from the current site's `reward` property.
  - Set default value of `reward` to `false`.

- Testing:
  - Build, assemble, and deploy EAR files to the production and publishing servers.
  - Verify that `OrderManager` properties are set correctly.

- Use Context link in **Dynamo Admin UI** to change context to **US Reward Site**; verify that `OrderManager.reward` is now `true`.
  - Open store in different browser. Navigate between **ATG Store** and **US Reward Site** to create a new order on both sites.
  - Use `/atg/dynamo/servlet/sessiontracking/GenericSessionManager` to view `/atg/commerce/ShoppingCart.current`. By using the Context link to navigate between site contexts, verify that the `reward` property is set to `false` on **ATG Store** and `true` on **US Reward Site**.
  - Log in to **US Reward Site** with test user `alex@example.com`. The password is `password`.

- Open Production **ACC** and verify that Alex's order has been persisted with `reward` set to `true`.
- Log out of the store.

You are encouraged to try to complete this exercise by using only the plan above (you will not need to create the database tables, as they have already been created for you). You should use the suggested variable, class, and component names.

Alternatively, there are step-by-step instructions starting on the next page if you prefer.

Guided Instruction

Task 1: Extend the order repository and order persistence.

1. The `mys_order` table has already been created for you in the `educommprod and educommpub` schemas of the Oracle database. View this table by using `SQL*Plus`.
2. Using a text editor or Eclipse, create a file named `/atg/commerce/order/orderrepository.xml` under the **`MyStore.Custom`** `config` folder. Add the following content to this file:

```
<gsa-template>
 <item-descriptor name="order">
  <table name="mys_order" type="auxiliary"
   id-column-name="order_id">

    <property name="reward" data-type="boolean"
     column-name="reward" category="My Store Extensions"
     default="false"  />

  </table>
 </item-descriptor>
</gsa-template>
```

3. Create another file named `/atg/commerce/order/processor/LoadOrderObject.properties` under the **`MyStore.Custom`** `config` folder with the following content:

```
loadProperties+=reward
```

4. To test your repository changes:
   - Open a terminal window and navigate to `/ae101/ATG10.1.1/home/bin`.
   - Issue the following command (all on one line):

     ```
     ./startSQLRepository -m MyStoreFakeXA -repository
     /atg/commerce/order/OrderRepository
     ```

     Resolve any errors or warnings before proceeding.
   - Close the terminal window.
5. Rebuild project, assemble the `atg_production.ear` file, and deploy to WebLogic. Restart the production server if it is not already running.
6. After the server has finished starting, open the **Dynamo Admin UI** page (`http://localhost:7103/dyn/admin`). Use the Component Browser to browse down to `/atg/commerce/order/OrderRepository`. **Tip:** Alternatively, you may select **Search** and enter `"OrderRepository"`. Select `/atg/commerce/order/OrderRepository` from the list of search results.
7. Next to the `order` item, click **See Property Descriptions**. Scroll down and verify that the `reward` property is displayed. The default value should be `false`.

8. In the Component Browser, browse to or search for `/atg/commerce/order/processor/LoadOrderObject`. Verify that its `loadProperties` property includes `reward`.

Task 2: Extend the order class.

9. In Eclipse, switch to the Java EE perspective and create a class named `MyStoreOrder` in the `com.mystore.order` package. The class should extend `atg.projects.store.order.StoreOrderImpl`.
10. Add the following methods to define the `reward` property. These methods use the `getProperty` and `setProperty` methods of the superclass to get and set the property value in the underlying `order` repository item.

```
public void setReward(boolean reward) {
 setPropertyValue("reward", Boolean.valueOf(reward));
}

public boolean isReward() {
 Boolean returnValue =
 (Boolean)getPropertyValue("reward");

 return (returnValue == null ? false : returnValue.booleanValue());
}
```

11. Save the class.
12. Still in Eclipse, switch back to the ATG perspective and create a file named `/atg/commerce/order/OrderTools.properties` under the **`MyStore.Custom`** `config` folder with the following content:

```
beanNameToItemDescriptorMap-=\
atg.projects.store.order.StoreOrderImpl=order

beanNameToItemDescriptorMap+=\
com.mystore.order.MyStoreOrder=order

orderTypeClassMap+=\
shoppingcart=com.mystore.order.MyStoreOrder,\
default=com.mystore.order.MyStoreOrder
```

13. Rebuild project, assemble the `atg_production.ear` file, and deploy to WebLogic.
14. Open the **Dynamo Admin UI** page and use the Component Browser to navigate to or search for `/atg/commerce/order/OrderTools`. Verify that the entry for `order` in the `beanNameToItemDescriptorMap` property points to your class. Verify that the entries in the `orderTypeClassMap` property also point to your class.
15. Open a new browser window and enter `http://localhost:7103/crs` in the URL.

16. Switch back to the **Dynamo Admin UI** browser page and use the Component Browser to navigate to `/atg/dynamo/servlet/sessiontracking/GenericSessionManager`. This page lets you view the session-scoped components for each session on the server. **Tip:** Bookmark this page for future use during the course.
17. Click the **View** button to list the current sessions. There will be at least two sessions: one for your admin client and one for the store.
18. Click the session IDs to view the components in those sessions. The session that includes an `/atg/commerce` folder is the store session.
19. In the store session, navigate to `/atg/commerce/ShoppingCart`. Scroll down to view the properties of the `ShoppingCart` component.
20. Click the `current` property, which represents the active order. Confirm that the class of the order is `com.mystore.order.MyStoreOrder` and that the `reward` property is listed.

Task 3: Extend the order manager.

21. In Eclipse, create a class named `MyStoreOrderManager` in the `com.mystore.order` package. The class should extend `atg.projects.store.order.StoreOrderManager`.
22. Create a new `boolean` property named `reward`, with the appropriate get and set methods (Tip: After declaring the member variable `reward`, click the **Source** menu and then **Generate Getters and Setters**).
23. Select the **Source** menu and then **Override/Implement Methods**.
24. In the pop-up window, select the **createOrder** method and click **OK**.
25. Eclipse will insert the method with generic variable names (arg0, and so on). Replace these names with more informative variable names to make the code easier to understand:

    arg0 = `pProfileId`
    arg1 = `pOrderId`
    arg2 = `pOrderPriceInfo`
    arg3 = `pTaxPriceInfo`
    arg4 = `pShippingPriceInfo`
    arg5 = `pOrderType`

    Note: Be sure to replace the variable names in both the method signature and the call to `super.createOrder()`.

26. Erase the comment generated by Eclipse (`// TODO Auto-generated method stub`).
27. Alter the call to `super.createOrder` from:

```
return super.createOrder(pProfileId, pOrderId, pOrderPriceInfo,
pTaxPriceInfo, pShippingPriceInfo, pOrderType);
```

    to:

```
MyStoreOrder order = (MyStoreOrder) super.createOrder(pProfileId,
pOrderId, pOrderPriceInfo, pTaxPriceInfo, pShippingPriceInfo,
```

```
pOrderType);
```

28. Add the following lines after the call to `super.createOrder`:

```
order.setReward(isReward());
return order;
```

29. Save the class.
30. Using a text editor, create a file named `/atg/commerce/order/OrderManager.properties` under the **`MyStore.Custom`** `config` folder with the following content:

```
$class=com.mystore.order.MyStoreOrderManager

$instanceFactory=\
/atg/multisite/SiteSourcedPropertyGetterSubClasser

$overridePropertyToValuePropertyMap=\
reward=reward

reward=false
```

31. Rebuild project, assemble the `atg_production.ear` file, and deploy to WebLogic.
32. In the **Dynamo Admin UI** page, navigate to `/atg/commerce/order/OrderManager`. Verify that the class of the component is `MyStoreOrderManager`.
33. Scroll down to verify that there is now a `reward` property. The value should be `false`.
34. To view how the value of the `reward` property would change based on the current site context, we can use the **Context** link at the top of the **Dynamo Admin UI** page. This link appears on every page, just below the header.

    Click the **Context** link. In the **Site** drop-down, select **US Reward Site**. Click the **Set** button.
35. You will be returned to the component page after setting the site. Note that the Context link now includes the current site context. Scroll down the `reward` property, which should now be `true`.
36. At the top of the page, click the **Clear All** link to clear the site context.
37. Open or refresh the home page of the store to start a new session.
38. In the **Dynamo Admin UI** page, navigate to `/atg/dynamo/servlet/sessiontracking/GenericSessionManager`.
39. Click the **View** button to list the current sessions and locate the store session. It will have the `/atg/commerce` folder.
40. In the store session, navigate to `/atg/commerce/ShoppingCart`.
41. Use the **Context** link to set the site context to **ATG Store**.
42. Click to `current` property to view the current order. Note the order's id. Confirm that the reward property is set to `false`.
43. Switch back to the store page. Navigate to the **US Reward Site**.
44. Switch to the **Dynamo Admin UI** page. Change the site context to **US Reward Site**. Click to view the current order again. The `reward` property should now be set to `true`. Note that this is a new order object with a different id. Make a note of this id for later.

45. To test the order persistence, go back to the store and log in as `alex@example.com` (the password for all the test users is `password`).
46. Open the Production **ACC** and navigate to the list of orders under **Purchases and Payments\Orders**. Click the **List** button again to refresh the list. Locate Alex's order by using the id from the previous step.
47. Select Alex's order and confirm that the `reward` property is `true`.
48. In the store, log out to return to shopping anonymously.

Practices for Lesson 10: Pricing

Chapter 10

Practices for Lesson 10

Practices Overview

There are no practices for this lesson.

Practices for Lesson 11: Displaying and Extending Pricing

Chapter 11

Practice 11-1: Extending Pricing

Overview

In this practice, you will be extending the pricing system. At the end of this exercise you will be able to:

- Extend the `OrderPriceInfo` class
- Create a custom pricing calculator for orders
- Add pricing calculators to the order pricing engine

Problem Statement

Shoppers should earn 1 point for each dollar they spend, excluding shipping and tax. For simplicity, we will calculate the number of points earned at the order level, based on the order's subtotal. (The disadvantage of this approach is that item-level points promotions are not supported.) Points will be stored in the order's `priceInfo` object.

For now, we will calculate the points based on the subtotal of the order before any discounts have been applied. In the next lab, you will extend `OrderDiscountCalculator` so that it adjusts the number of points whenever it discounts the order subtotal.

In order to test the purchasing of items on the **US Reward Site**, we will need to fill in a price for at least one item in its price list.

Planning

Take a moment before looking at the next page to think about what you will need to do to accomplish the tasks discussed above, as you have in the previous labs. A completed plan begins on the next page.

Plan

Task 1: Setting prices for reward items

- Use the Merchandising section of the **BCC** to set prices for
  - Paris Clock
  - Huey Martini Glass

Task 2: Extending the `OrderPriceInfo` repository item

- Create new table called `mys_order_price` in production and publishing data sources.
  - The new table will have two columns: `amount_info_id` (`varchar2(40)`) and `points` (`number(10)`).

- Extend `orderrepository.xml` under **`MyStore.Custom`**.
  - Item descriptor to extend is `orderPriceInfo`
  - Table is auxiliary.
  - `points` is an `int` property.

- Alter the `/atg/commerce/order/processor/LoadPriceInfoObjects` component.
  - Add `points` to its `loadProperties` property.

- Alter the `/atg/commerce/order/processor/SavePriceInfoObjects` component.
  - Add `points` to its `savedProperties` property.

- Testing:
  - Use `startSQLRepository` to test extension to `orderPriceInfo`.
  - Build, assemble, and deploy EAR files to the production and publishing servers.
  - View Order Repository through Component Browser to verify that `OrderPriceInfo` has a `points` property.
  - View `LoadPriceInfoObjects` and `SavePriceInfoObjects` and verify that their properties are correct.

Task 3: Extending the `OrderPriceInfo` class

- Create new class called `com.mystore.pricing.MyStoreOrderPriceInfo`.
  - Extend `atg.commerce.pricing.OrderPriceInfo`.
  - Add get and set methods for an `int` property `points`.

- Alter the properties of `OrderTools` so that new order price info class is used.
  - Remove `atg.commerce.pricing.OrderPriceInfo`, with `key orderPriceInfo`, from the `beanNameToItemDescriptorMap` property.
  - Add `com.mystore.pricing.MyStoreOrderPriceInfo` to the `beanNameToItemDescriptorMap` property, with key `order.priceInfo`.

- Alter the properties of `OrderPricingEngine` so that new order price info class is used.
  - Change value of `priceInfoClass` to `com.mystore.pricing.MyStoreOrderPriceInfo`.

- Testing:
  - Build, assemble, and deploy EAR files to the production and publishing servers.
  - In **ATG Store**, start a new session.
  - Use `GenericSessionManager` to view the current order's price information. Verify that it is using the correct class.

Task 4: Creating and configuring the pricing calculator

- Create new class called `com.mystore.pricing.MyStoreOrderInitialPointsCalculator`.
  - Extend `atg.nucleus.GenericService`
  - Implement `atg.commerce.pricing.OrderPricingCalculator`
  - Import `com.mystore.order.MyStoreOrder`
  - In `priceOrder`:
    - Cast the order as `MyStoreOrder`
    - If the `reward` property of the order is **not** true:
      - Get the `amount` from `pPriceQuote`.
      - Cast `pPriceQuote` as `MyStoreOrderPriceInfo`.
      - Set `points` property of price info equal to `amount`, converted to an `int`.

- Create new component named `/com/mystore/pricing/OrderInitialPointsCalculator`.
  - Based on `com.mystore.pricing.MyStoreOrderInitialPointsCalculator`

- Alter `OrderPricingEngine` to use new calculator.
  - Add `/com/mystore/pricing/OrderInitialPointsCalculator` to the end of the `preCalculators` property.

- Testing:
  - Build, assemble, and deploy EAR files to the production and publishing servers.
  - Verify that `OrderPricingEngine.preCalculators` is set correctly.
  - Log in to **ATG Store** with test user `lisa@example.com`. Password is `password`. Place an order for the Crystal Decanter.
  - Navigate to the **US Reward Site** and place an order for the Paris Clock.
  - Open the Production **ACC** and verify that both Lisa's orders have been placed and in the submitted state.
  - Verify that the order placed on the **ATG Store** has the correct points in the order price info object and the order placed on the **US Rewards Site** has `0` points in the order price info object.

You are encouraged to try to complete this exercise by using only the plan above (you will not need to create the database tables, as they have already been created for you). You should use the suggested variable, class, and component names.

Alternatively, there are step-by-step instructions starting on the next page if you prefer.

Guided Instruction

Task 1: Setting prices for reward items

1. Start your **Publishing Server** if it is not running.
2. Open the **BCC** and log in as `admin/password1`.
3. Expand the **Merchandising** section of the Operations menu and click **Manage Commerce Assets**.
4. Name the project as `Set Reward Prices` and click **Continue**.
5. Navigate down through **Site Catalogs > US Reward Site.**
6. Expand the `create asset (+)` dropdown and select Catalog Folder.
7. Enter "`US Reward Catalog Folder`" in the name and select the Master Catalog for the Child Catalog property.
8. Click **Create** to create the Catalog Folder.
9. Navigate down through **US Reward Catalog Folder > Master Catalog > Commerce Root > Home Accents > Clocks > Paris Clock**.
10. You should now see the single SKU for Paris Clock. Double-click this SKU to open it for editing.
11. On the right, navigate to the **Price Lists** tab. In the **Price Lists** field, select **Reward US**. Change the **Price Type** from **Inherit** to **Override**. Fill in a **SKU Price** of `100`. Click the **Save** button.
12. On the left, navigate back up to the **Home Accents** category and then down to **Glassware > Huey Martini Glass**. You should now see the single SKU for Huey Martini Glass. Repeat the previous step to give it a price of `25`.
13. In the top middle of the screen, click the drop-down next to **Author** and select **Ready for Review**. In the pop up confirmation box, click **Continue**.
14. The project stage has now changed to **Content Review**. Click the drop-down next to the stage name and select **Approve Content**. Again, click **Continue** in the pop up confirmation box.
15. The project stage has now changed to **Approve for Production Deployment**. Click the drop-down next to the stage name and select **Approve and Deploy to Production**. Again, click **Continue** in the pop up confirmation box. Click the **Home** button in the following pop up.
16. If you want, switch to the **CA server** console to view the progress of the deployment. When the deployment is finished, you will see a message similar to:

```
**** info       Fri Dec 14 12:59:43 EST 2012    1355507983724
/atg/deployment/DeploymentManager       Purging deployment data
for deployment 1900003
**** info       Fri Dec 14 12:59:43 EST 2012    1355507983788
/atg/deployment/DeploymentManager       Deployment 1900003
finished in 0:09.235
```

17. Open the **US Reward Site** in a browser. Navigate to the Paris Clock to confirm its price is set correctly to $100 (**Tip**: The clock is one of the featured products in the **Home Accents** category). Repeat for the Huey Martini Glass. In the practice titled "Extending Payment Groups" you will create a processor to make purchases using points.

Task 2: Extending the `OrderPriceInfo` repository item

18. The `mys_order_price` table has already been created for you in the `educommprod` and `educommpub` schemas of the Oracle database. View this table by using `SQL*Plus`.
19. In a text editor or Eclipse, open the `/atg/commerce/order/orderrepository.xml` file that you created in the previous lab. Add the following just above the `</gsa-template>` tag:

```
<item-descriptor name="orderPriceInfo">
 <table name="mys_order_price" type="auxiliary"
  id-column-name="amount_info_id">

  <property name="points" data-type="int"
  column-name="points" />

 </table>
</item-descriptor>
```

20. To test your changes:
    - Open a command prompt and navigate to `<atgdir>/home/bin`.
    - Issue the following command (all on one line):

      ```
      startSQLRepository -m MyStoreFakeXA -repository
      /atg/commerce/order/OrderRepository
      ```

      Resolve any errors before proceeding.
    - Close the command window.
21. Using a text editor or Eclipse, create a file named `/atg/commerce/order/processor/LoadPriceInfoObjects.properties` under the **`MyStore.Custom`** `config` folder with the following content:

    ```
    loadProperties+=points
    ```

22. Using a text editor or Eclipse, create a file named `/atg/commerce/order/processor/SavePriceInfoObjects.properties` under the **`MyStore.Custom`** `config` folder with the following content:

    ```
    savedProperties+=points
    ```

23. Rebuild project, assemble the `atg_production.ear` file, and deploy to WebLogic.
24. Use the **Dynamo Admin UI** to confirm that the `points` property is configured for the `OrderPriceInfo` repository item (browse to the `/atg/commerce/order/OrderRepository` component and click **See Property Descriptions** for `OrderPriceInfo`).
25. Browse to `LoadPriceInfoObjects` and confirm that the `loadProperties` property includes `points`. Do the same for `SavePriceInfoObjects.savedProperties`.

Task 3: Extending the `OrderPriceInfo` class

26. In Eclipse, switch to the Java EE perspective and create a class named `MyStoreOrderPriceInfo`. This class should be in the `com.mystore.pricing` package and extend `atg.commerce.pricing.OrderPriceInfo`.
27. Add a private member `int` variable named `points` and the get and set methods for it.
28. Save the class. Watch for errors in the Problems tab.
29. Using a text editor, open the `/atg/commerce/order/OrderTools.properties` file that you created in the **`MyStore.Custom`** `config` folder in an earlier lesson. Change the text that reads:

    ```
    beanNameToItemDescriptorMap-=\
    atg.projects.store.order.StoreOrderImpl=order
    ```

    to (additions in bold)

    ```
    beanNameToItemDescriptorMap-=\
    atg.projects.store.order.StoreOrderImpl=order,\
    atg.commerce.pricing.OrderPriceInfo=orderPriceInfo
    ```
30. Change the text that reads:

    ```
    beanNameToItemDescriptorMap+=\
    com.mystore.order.MyStoreOrder=order
    ```

    to (additions in bold)

    ```
    beanNameToItemDescriptorMap+=\
    com.mystore.order.MyStoreOrder=order,\
    com.mystore.pricing.MyStoreOrderPriceInfo=\
    orderPriceInfo
    ```

31. Create a file named `/atg/commerce/pricing/OrderPricingEngine.properties` in the **`MyStore.Custom`** `config` folder with the following content:

    ```
    priceInfoClass=com.mystore.pricing.MyStoreOrderPriceInfo
    ```
32. Rebuild project, assemble the `atg_production.ear` file, and deploy to WebLogic.
33. Open the store to begin a new session. Use the `GenericSessionManager` to confirm that the class for the new session's order's `priceInfo` object is your class, and that the `points` property is there.

Task 4: Creating and configuring the pricing calculator

34. In Eclipse, switch to the Java EE perspective and create a class named `MyStoreOrderInitialPointsCalculator` in the `com.mystore.pricing` package. This class should extend `atg.nucleus.GenericService` and implement `atg.commerce.pricing.OrderPricingCalculator`.

35. Add the following import statement:

```
import com.mystore.order.MyStoreOrder;
```

36. Erase the comment that Eclipse has automatically generated (`// TODO Auto-generated method stub`).
37. Change the generic variable names in the `priceOrder` method to more informative names:

```
public void priceOrder (OrderPriceInfo pPriceQuote, Order pOrder,
RepositoryItem pPricingModel, Locale pLocale, RepositoryItem
pProfile, Map pExtraParameters)
```

38. In the body of the `priceOrder` method, add the following code:

```
MyStoreOrder myOrder = (MyStoreOrder) pOrder;

if (!myOrder.isReward()){
 int points = (int) pPriceQuote.getAmount();
 MyStoreOrderPriceInfo myInfo = (MyStoreOrderPriceInfo)
  pPriceQuote;
 myInfo.setPoints(points);
} //end if not reward order
```

39. In Eclipse, switch to the ATG perspective and create a new component named `/com/mystore/pricing/OrderInitialPointsCalculator`. The class of this component should be `com.mystore.pricing.MyStoreOrderInitialPointsCalculator`.
40. Still in Eclipse, create a file named `/atg/commerce/pricing/OrderPricingEngine.properties` under the **`MyStore.Custom`** `config` folder with the following content:

```
priceInfoClass=com.mystore.pricing.MyStoreOrderPriceInfo
preCalculators+=/com/mystore/pricing/OrderInitialPointsCalculato
r
```

41. Rebuild project, assemble the `atg_production.ear` file, and deploy to WebLogic.
42. In the **Dynamo Admin UI**, confirm that `OrderPricingEngine.priceInfoClass` and `OrderPricingEngine.preCalculators` are set correctly.
43. . Return to the **ATG Store**. For the first order, you want to be on the **ATG Store** site. If you are looking at the **US Reward Site**, click the link to switch to **ATG Store**. Log in as `lisa@example.com`.
44. Add a Crystal Decanter to the shopping cart.
45. Navigate to the shopping cart and click **Checkout** to begin the checkout process.
46. Select the saved shipping address and click **Ship to this Address**. Select the default shipping method and click **Continue**.
47. Enter `"123"` for the CSV and click **Continue**.
48. On the order review page, click **Place My Order** and note the order number on the next page.

49. Navigate to the **US Rewards Site** and repeat steps 44-48 with the Paris Clock. In this practice, you purchase the clock with a credit card. In a later practice, you will create a custom payment group to purchase with points.
50. Open the **ACC** and navigate to **Purchases and Payments > Orders**. Click the **List** button and locate both Lisa's orders by using the IDs you noted earlier.
51. Double-click the **Price info** property for each order to view the price info repository item. Note the **points** property for each of the orders and their values. The order placed on the **ATG Store** should have received `41` points and the order placed on the **US Reward Site** should have received `0` points.
52. Log out of the store to return to shopping anonymously.

Practices for Lesson 12: Extending Order Discount Calculators

Chapter 12

Practice 12-1: Order Discount Calculators

Overview

Practices titled “Extending Order Discount Calculators,” “Custom Promotion Calculator,” and “Creating a Custom Promotion Template” cover extending promotions. In this practice, you will be extending promotions to adjust points awarded based on items that qualify instead of the order total. In the following practices, you will be creating a custom calculator and promotion. At the end of this exercise you will be able to:

- Extend the `OrderDiscountCalculator` class

Problem Statement

In the last lab, you created a calculator that calculated the points for an order before any discounts had been applied, which meant that the points amount might have been more than the final subtotal. The existing order discount calculators will need to be extended to adjust the points property whenever they adjust the subtotal.

Points promotions should not run on reward orders, because they can never earn points. To improve performance, points promotions will be eliminated from being evaluated on reward orders.

Planning

Take a moment before looking at the next page to think about what you will need to do to accomplish the tasks discussed above, as you have in the previous labs. A completed plan begins on the next page.

Plan

Task: Extending the existing order discount calculators

- Create new class called `com.mystore.pricing.MyStoreOrderDiscountCalculator`.
  - Extend `atg.commerce.pricing.OrderDiscountCalculator`
  - Import `com.mystore.order.MyStoreOrder`
  - Override `priceOrder()`
    - Call `super.priceOrder()`
    - Add code to reprice points amount

- Create a new class called `com.mystore.pricing.MyStoreBulkOrderDiscountCalculator`.
  - Extend `atg.commerce.pricing.BulkOrderDiscountCalculator`
  - Import `com.mystore.order.MyStoreOrder`
  - Override `priceOrder()`
    - Call `super.priceOrder()`
    - Add code to reprice points amount

- Alter existing discount calculator components to use new classes.
  - Change the class of `/atg/commerce/pricing/calculators/OrderDiscountCalculator` to `com.mystore.pricing.MyStoreOrderDiscountCalculator`.
  - Change the class of `/atg/commerce/pricing/calculators/BulkOrderDiscountCalculator` to `com.mystore.pricing.MyStoreBulkOrderDiscountCalculator`.

- Testing:
  - Build, assemble, and deploy EAR files to the production and publishing servers.
  - Verify that the order discount calculators are using the correct classes.
  - In the **ATG Store**, log in as lisa@example.com, and add a Paris Clock to the order.
  - Apply the `TENSHIP` coupon on the checkout page.
  - Submit the order and note the order id.
  - Locate the order in the **ACC** and look for the number of points rewarded that should have been adjusted for the promotion applied.

You are encouraged to try to complete this exercise by using only the plan above. You should use the suggested variable, class, and component names.

Alternatively, there are step-by-step instructions starting on the next page if you prefer.

Guided Instruction

Task: Extending the existing order discount calculators

1. In Eclipse, switch to the Java EE perspective and create a new class named `MyStoreOrderDiscountCalculator` in the `com.mystore.pricing` package. The class should extend `atg.commerce.pricing.OrderDiscountCalculator`.
2. Add the following import statement:

   ```
   import com.mystore.order.MyStoreOrder;
   ```
3. Select the **Source** menu and then **Override/Implement Methods**.
4. In the pop-up window, select the **priceOrder** method and click **OK**.
5. Eclipse will insert the method with generic variable names (arg0 and so on). Replace these with more informative variable names:

   arg0 = `pPriceQuote`
   arg1 = `pOrder`
   arg2 = `pPricingModel`
   arg3 = `pLocale`
   arg4 = `pProfile`
   arg5 = `pExtraParameters`

   Note: Make sure to replace the variable names in both the method signature and the call to `super.priceOrder()`.

6. Erase the comment generated by Eclipse (`// TODO Auto-generated method stub`).
7. Add the following lines after the call to `super.priceOrder` (this is the same code from `MyStoreOrderInitialPointsCalculator` in the previous chapter):

   ```
   MyStoreOrder myOrder = (MyStoreOrder) pOrder;

   if (!myOrder.isReward()){
    int points = (int) pPriceQuote.getAmount();
    MyStoreOrderPriceInfo myInfo = (MyStoreOrderPriceInfo)
     pPriceQuote;
    myInfo.setPoints(points);
   } //end if not reward order
   ```

8. Save the class, build project, and watch for errors in the error log tab.
9. Create a new class named `MyStoreBulkOrderDiscountCalculator` in the `com.mystore.pricing` package. The class should extend `atg.commerce.pricing.BulkOrderDiscountCalculator`.
10. Repeat Steps 2-8 for this class. **Note:** To select the `priceOrder` method for overriding, you will need to expand **OrderDiscountCalculator**.

11. Create a properties file named `/atg/commerce/pricing/calculators/OrderDiscountCalculator.properties` under the **`MyStore.Custom`** `config` folder with the following content:
    `$class=com.mystore.pricing.MyStoreOrderDiscountCalculator`
12. Create another properties file named `/atg/commerce/pricing/calculators/BulkOrderDiscountCalculator.properties` under the **`MyStore.Custom`** `config` folder with the following content:
    `$class=com.mystore.pricing.MyStoreBulkOrderDiscountCalculator`
13. Rebuild project, assemble the `atg_production.ear` file, and deploy to WebLogic.
14. In the **Dynamo Admin UI**, confirm that the classes for `OrderDiscountCalculator` and `BulkOrderDiscountCalculator` are set correctly.
15. Go to the **ATG Store** and log in as `lisa@example.com`. Add a Paris Clock to the shopping cart.
16. In the **ATG Store**, click the **Checkout** link at the upper right corner. In the **Order Summary** section of the page, locate the **Coupon Code** field. Enter TENSHIP in this field and click **Apply Coupon** (the TENSHIP coupon grants a promotion that takes 10% off the order total and free shipping).
17. Continue the checkout process to submit the order and note the order id.
18. Launch the **ACC**, search for the order, and verify that the points granted are adjusted for the amount of the promotion.

Practices for Lesson 13: Custom Promotion Calculator

Chapter 13

Practice 13-1: Custom Calculators

Overview

In this practice, you continue extending promotions by creating a calculator for a custom promotion type. At the end of this exercise you will be able to:

- Create a custom calculator

Problem Statement

The store will also implement a custom promotion type that increases an order's points by a certain multiplier (for example, double points in the month of December).To support this promotion type, you will need to create a custom calculator that calculates total points earned based on the promotion.

Planning

Take a moment before looking at the next page to think about what you need to do to accomplish the tasks discussed above, as you have in the previous labs. A completed plan begins on the next page.

Plan

Task: Creating a custom promotion calculator

- Create a new class called `com.mystore.points.MyStoreOrderPointsPromotionCalculator`.
  - Extend `atg.nucleus.GenericService`
  - Implement `atg.commerce.pricing.OrderPricingCalculator`
  - Import:
    - `atg.commerce.pricing.Constants`
    - `atg.commerce.pricing.definition.DiscountStructure`
    - `atg.commerce.pricing.definition.MatchingObject`
    - `atg.commerce.pricing.PricingAdjustment`
    - `com.mystore.pricing.MyStoreOrderPriceInfo`
  - In the `priceOrder()` method:
    - Retrieve `MatchingObject` and `DiscountStructure` from `pExtraParameters`.
    - Get `adjuster` from `DiscountStructure`.
    - Get `quantity` from `MatchingObject`.
    - Get `pPriceQuote` as a `MyStorePricingInfo` object.
    - Create two `int` variables, `newPoints` and `oldPoints`, both set to the current `points` amount.
    - Create a for loop based on `MatchingObject.quantity`.
      - Each time through the loop, multiply the `points` amount by the adjuster and set `newPoints` to the result.
    - Set `points` value in price info object from `newPoints`.
    - Calculate the difference between `newPoints` and `oldPoints`.
    - Create the `PricingAdjustment` object with the following constructor arguments:
      - `"OrderPointsMultiplier"`
      - `pPricingModel`
      - adjustment amount
      - 1
    - Add new `PricingAdjustment` to `priceInfo.adjustments`.
    - Return the price info object.

- Create new component named `/com/mystore/points/MyStoreOrderPointsPromotionCalculator`.
  - Based on `com.mystore.points.MyStoreOrderPointsPromotionCalculator`

- Alter `OrderPricingEngine` to use new calculator.

- Add `/com/mystore/points/MyStoreOrderPointsPromotionCalculator` to `calculatorTypeCalculators property` with a key of `points`.

- Testing:
  - Build, assemble, and deploy `production.ear` to the production server.
  - Verify that `OrderPricingEngine.CalculatorTypeCalculators` is set correctly.

You are encouraged to try to complete this exercise by using only the plan above. You should use the suggested variable, class, and component names.

Alternatively, there are step-by-step instructions starting on the next page if you prefer.

Guided Instruction

Task: Creating a custom promotion calculator

1. In Eclipse, create a new class named `MyStoreOrderPointsPromotionCalculator` in the `com.mystore.points` package. The class should extend `atg.nucleus.GenericService` and implement `atg.commerce.pricing.OrderPricingCalculator`.
2. Add the following import statements:
```
import atg.commerce.pricing.Constants;
import atg.commerce.pricing.definition.DiscountStructure;
import atg.commerce.pricing.definition.MatchingObject;
import atg.commerce.pricing.PricingAdjustment;
import com.mystore.pricing.MyStoreOrderPriceInfo;
```
3. Erase the comment Eclipse has automatically generated (`// TODO Auto-generated method stub`).
4. Change the generic variable names in the `priceOrder` method to more informative names:
```
public void priceOrder (OrderPriceInfo pPriceQuote, Order pOrder,
RepositoryItem pPricingModel, Locale pLocale, RepositoryItem
pProfile, Map pExtraParameters)
```
5. Add the following code to the `priceOrder` method:
```
MatchingObject matchingObject = (MatchingObject)
 pExtraParameters.get(Constants.MATCHING_OBJECT);

DiscountStructure discountStructure = (DiscountStructure)
 pExtraParameters.get(Constants.DISCOUNT_STRUCTURE);

Double adjuster = discountStructure.getAdjuster();

MyStoreOrderPriceInfo priceInfo =
 (MyStoreOrderPriceInfo) pPriceQuote;

int newPoints = priceInfo.getPoints();
int oldPoints = priceInfo.getPoints();

//may qualify more than once if "for next" rule is used

for (int i = 0; i < matchingObject.getQuantity(); i++) {
  newPoints = (int) (newPoints * adjuster);
} //end for all the times order qualifies

double adjustAmount = newPoints - oldPoints;
```

```
priceInfo.getAdjustments().add (new
PricingAdjustment("OrderPointsMultiplier", pPricingModel,
adjustAmount, 1));

priceInfo.setPoints(newPoints);
```

6. Build project and switch to the ATG perspective and the ATG Component Browser. Create a new component under `/com/mystore/points` named `MyStoreOrderPointsPromotionCalculator`. The component should be based on the `com.mystore.points.MyStoreOrderPointsPromotionCalculator` class.

7. Switch to the Package Explorer tab and open the `/atg/commerce/pricing/OrderPricingEngine.properties` file under the **`MyStore.Custom`** `config` folder for editing. Add the following lines:

```
calculatorTypeCalculators+=\
points=\
/com/mystore/points/MyStoreOrderPointsPromotionCalculator
```

8. Rebuild project, assemble the `atg_production.ear` file, and deploy to WebLogic.
9. In the **Dynamo Admin UI**, verify that `OrderPricingEngine.calculatorTypeCalculators` includes your calculator.

Practices for Lesson 14: Creating a Custom Promotion Template

Chapter 14

Practice 14-1: Custom Promotion and Template

Overview

In this practice, you will be creating a custom promotion template to create a points based promotion. At the end of this exercise you will be able to:

- Create a custom promotion template
- Create a custom promotion

Problem Statement

In the previous two lessons, you created a custom order discount calculator and a custom promotion calculator for a new points-based promotion type. The new promotion type also requires a new interface to create the custom promotion. Any text shown in the promotion template should be internationalizable.

Planning

Take a moment before looking at the next page to think about what you will need to do to accomplish the tasks discussed above, as you have in the previous labs. A completed plan begins on the next page.

Plan

Task: Creating a custom promotion template

- Create new resource bundle under **`MyStore.Custom`** for new promotion template labels.
  - Name the resource bundle as `com.mystore.points.PromotionTemplateResources`.
  - Add the following keys and values:

```
template.pointsmultipler.segmentConditionTitle
=Condition

template.pointsmultipler.applyOfferLabel
=Multiply points under these conditions:

template.pointsmultiplier.title=Multiply Points

template.pointsmultiplier.segmentOfferTitle=Offer

template.pointsmultiplier.multiplerNumber
=Number by which to multiply points:

template.pointsmultiplier.multiplerNumberHelp
=e.g. to double the points, enter 2
```

- Create a new promotion template.
  - The template is named `/atg/registry/data/promotiontemplates/mystore/PointsMultiplier.pmdt`
  - Copy header and "condition" screen segment from `DCS/config/atg/registry/data/promotiontemplates/advanced/advancedOrderDiscount.pmdt`.
    - Change `display-resource` values to point to the new resource keys.
    - Change `available-in-ui` value to `true`.
  - Add an `offer` screen segment.
    - Add a label for multiplier number field.
    - Add a text field to collect multiplier number.
    - Add a label after field for field help.
  - Copy the `multi-element-translators` section from `advancedOrderDiscount.pmdt`.
  - Copy the `item-properties` section from `advancedOrderDiscount.pmdt`.
    - Open the `TENSHIP` promotion in the **BCC** through the **Content Administration\Browse assets** view to see the raw PMDL.

        - In the template, replace `${discount-structure}` with the contents of the offer tag.
        - Alter the copied discount-structure to replace:
            - `adjuster="2.0"` with `adjuster="${multiplier}"`
            - `discount-type="fixedPrice"` with `discount-type=""`
            - `calculator-type="points"`

- Testing:
    - Build, assemble, and deploy EAR files to the production and publishing servers.
    - Watch for errors from the `PromotionTemplateManager` on startup. Resolve any errors, rebuild, and restart if necessary.
    - Start a new project to create a `Double points` promotion.
        - Change to use your new template.
        - Deploy the project to production.
    - In the store, log in as `lisa@example.com` and create an order.
    - In the **ACC**, verify that the order received double points.

You are encouraged to try to complete this exercise by using only the plan above. You should use the suggested variable, class, and component names.

Alternatively, there are step-by-step instructions starting on the next page if you prefer.

Guided Instruction

Task: Creating a custom promotion template

1. In Eclipse, create a file named `/com/mystore/points/PromotionTemplateResources.properties` under the **`MyStore.Custom`** `src` folder (not in the `config` folder).
2. Add the following content to this file:

```
template.pointsmultipler.segmentConditionTitle=Condition

template.pointsmultipler.applyOfferLabel=Multiply points under
these conditions:

template.pointsmultiplier.title=Multiply Points

template.pointsmultiplier.segmentOfferTitle=Offer

template.pointsmultiplier.multiplerNumber=Number by which to
multiply points:

template.pointsmultiplier.multiplerNumberHelp=e.g. to double the
points, enter 2
```

(**Note:** Each key-value pair should be on a single line)

3. Using a text editor or Eclipse, create a file named `/atg/registry/data/promotiontemplates/mystore/PointsMultiplier.pmdt` under the **`MyStore.Custom`** `config` folder.
4. Add the following content to the file:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE template PUBLIC
"-//Art Technology Group, Inc.//DTD PMDT 1.0//EN"
"http://www.atg.com/dtds/pmdt/pmdt_1.0.dtd">

<template item-type="Order Discount">

<ui-description available-in-ui="true"
 display-name-resource="template.pointsmultiplier.title"
 resource-bundle=
 "com.mystore.points.PromotionTemplateResources">

  <screen-segment display-name-resource=
   "template.pointsmultipler.segmentConditionTitle">
    <line>
      <label id="apply_offer_label"
       display-name-resource=
       "template.pointsmultipler.applyOfferLabel"/>
```

```
        <expression id="conditionEditor"
         model-path=
         "/atg/remote/promotion/expreditor/
          PromotionConditionsExpressionModel"
```
(**Note:** The model path value should be on a single line.)
```
         required="false"/>
    </line>
  </screen-segment>

  <screen-segment display-name-resource=
   "template.pointsmultiplier.segmentOfferTitle">
    <line>
      <label id="multiplierNumber"
       display-name-resource=
       "template.pointsmultiplier.multiplerNumber"/>

      <textInput id="multiplierNumber_textInput"
       restrict="0-9" placeholder-name="multiplier"
       validator="/atg/remote/promotion/template/
       validators/NonZeroDecimalValidator"/>
```
(**Note:** The validator value should be on a single line.)
```
      <label id="multiplierNumberHelp"
       display-name-resource=
       "template.pointsmultiplier.multiplerNumberHelp"/>
    </line>
  </screen-segment>

  <multi-element-translators>
    <multi-element-translator id="qualifier_translator"
     translator-path="/atg/remote/promotion/template/
      translators/QualifierTranslator">
```
(**Note:** The translator path value should be on a single line.)
```
      <placeholder-info placeholder-name="qualifier"
       translator-output-name="qualifier"/>
    </multi-element-translator>
  </multi-element-translators>
</ui-description>

<item-properties>
  <property name="pmdlRule">
    <![CDATA[
      <?xml version="1.0" encoding="UTF-8"
       standalone="no"?>
      <!DOCTYPE pricing-model SYSTEM
      "dynamosystemresource:/atg/dtds/pmdl/pmdl_1.0.dtd">

      <pricing-model>
        ${qualifier}
        <offer>
```

```
            <discount-structure adjuster="${multiplier}"
             calculator-type="points" discount-type=""/>
          </offer>
        </pricing-model>
      ]]>
    </property>
  </item-properties>
  </template>
```

5. Rebuild production and publishing EAR files, deploy, restart both servers, and watch for errors from `PromotionTemplateManager` (near the end of the startup process). Resolve any errors, and rebuild and restart the servers if necessary.
6. Open the **BCC** (login is `admin/password1`). In the **Operations** menu, expand the **Merchandising** section and click **Manage Commerce Assets**.
7. Name the new project `Double points` and click **Continue**.
8. Navigate down through **Promotions & Coupons > Promotions**.
9. Click the **New Item** button and select **Order Discount**.
10. Enter `Double points` in the new promotion's **Promotion name** field.
11. Select **Multiple Points** in the **Template** field.
12. In the **Condition and offer** section, leave the **Multiple Points** field set to `Always` and set the **Multiplier** to `2`.
13. Change to the **Availability & Coupons** tab. Click to edit the **Promotion availability** field. Select **Available to everyone**.
14. Click the **Create** button at the bottom of the screen to save the new promotion.
15. In the top middle of the screen, click the drop-down next to **Author** and select **Ready for Review**. In the pop-up confirmation box, click **Continue**.
16. The project stage has now changed to **Content Review**. Click the drop-down next to the stage name and select **Approve Content**. Again, click **Continue** in the pop-up confirmation box.
17. The project stage has now changed to **Approve for Production Deployment**. Click the drop-down next to the stage name and select **Approve and Deploy to Production**. Again, click **Continue** in the pop-up confirmation box. Click the **Home** button in the following pop-up.
18. Switch to the **CA console** window to verify that deployment was successful.
19. In the **ATG Store**, log in as `lisa@example.com` and place an order for a Crystal Decanter.
20. In the **ACC**, look up the order and verify that the customer received double points.
21. In the **BCC**, select to view All Projects and Tasks. Locate the `New double points` promotion and click to expand it. Click the **Verify Production Deployment** link.
22. In the navigation bar, click the project stage drop-down and select **Verify Production Deployment**. Click **Continue** in the pop-up confirmation box. Click the **Home** button in the following pop-up.

Practices for Lesson 15: Creating a Custom Shipping Group

Chapter 15

Practice 15-1: Customizing Shipping Groups

Overview

Lessons 15 and 16 will cover the shipping capability. In lesson 15, you will create a custom shipping type. In lesson 16, you will create a custom shipping method for the new type. In this practice, you will extend commerce to have a ship-to-store feature. At the end of this exercise you will be able to:

- Create a custom shipping group
- Create a shipping group initializer
- Display new shipping group in shipping page

Problem Statement

The store will have a ship-to-store function that allows shoppers to ship items to the store linked to their My Home Store profile property (created in the Section 3 lab).

To implement the ship-to-store functionality, you will create a new hardgood initializer component to initialize the ship-to-store shipping group from the user's profile information.

Planning

Take a moment before looking at the next page to think about what you will need to do to accomplish the tasks discussed above, as you have in the previous labs. A completed plan begins on the next page.

Plan

Task 1: Creating a Custom Shipping Group Type

- Create table called `mys_store_ship_group` in production and publishing datasources.
  - The new table will have two columns: `ship_group_id` (`varchar2(40)`) and `store_id` (`varchar2(40)`).

- Create class called `com.mystore.order.ShipToStoreShippingGroup`.
  - Extends `atg.commerce.order.HardgoodShippingGroup`.
  - Add get and set methods for `RepositoryItem store` property.
    - Methods should use `getPropertyValue()` and `getPropertyValue()`.

- Extend `orderRepository.xml` under **`MyStore.Custom`**.
  - Extend item descriptor `shippingGroup`.
    - Property to extend is `type`.
    - Add `option` tag with value `shipToStoreShippingGroup` and code `300`.
  - Create new item descriptor is named `shipToStoreShippingGroup`.
    - `super-type` is `hardgoodShippingGroup`.
    - Table is `mys_store_ship_group`.
    - Table is auxiliary.
    - Property is `store`, pointing to `store` item type in `/atg/store/stores/StoreRepository`.

- Alter properties of `OrderTools`.
  - Add `com.mystore.order.ShipToStoreShippingGroup` to `beanNameToItemDescriptorMap` property, with key `shipToStoreShippingGroup`.
  - Add `com.mystore.order.ShipToStoreShippingGroup` to property `shippingTypeClassMap`, with key `shipToStoreShippingGroup`.
  - Add `atg.commerce.order.RepositoryContactInfo` to property `shippingGroupShippingAddressClassNameMap`, with key `shipToStoreShippingGroup`.

- Testing:
  - Use `startSQLRepository` to test extension to `order`.
  - Build, assemble, and deploy an EAR file to the production server.
  - Verify that `OrderTools` properties are set correctly.
  - Verify that `shipToStoreShippingGroup` item descriptor exists in Order Repository.

Task 2: Creating a Custom Shipping Group Initializer

- Create new class called `com.mystore.order.purchase.MyStoreShipToStoreShippingGroupInitializer`.
  - Extends `atg.nucleus.GenericService`
  - Implements `atg.commerce.order.purchase.ShippingGroupInitializer` and `atg.commerce.order.purchase.ShippingGroupMatcher`
  - Add four properties:
    - `String storePropertyName`
    - `ShippingGroupManager shippingGroupManager`
    - `String shippingGroupType`
    - `String shippingMethod`
  - Implement the `initializeShippingGroups` method.
    - Get `store` from profile by using the property name configured in `storePropertyName`. (**Tip:** Use `DynamicBeans.getSubPropertyValue()`).
    - Create a new shipping group by using `ShippingGroupManager`.
    - Get the new shipping group's `shippingAddress` (class of address is `atg.commerce.order.RepositoryContactInfo`).
    - Set the shipping group's `shippingMethod` property from the value configured in the class' `shippingMethod` property.
    - Set the shipping group's `store` property to store.
    - Set the shipping group's address properties from store address values.
    - Set the shipping group's name properties from profile values.
    - Add shipping group to the input parameter `pShippingGroupMapContainer`.
  - Implement `getNewShippingGroupName` method.
    - If the shipping group is not ship-to-store, return null.
    - Otherwise, return the value of the shipping group's `shippingAddress'` `companyName` property.
  - Implement `matchShippingGroup` method.
    - If the shipping group is not ship-to-store, return null.
    - Otherwise, get `shippingGroupNames` from `pShippingGroupMapContainer`.
    - Get the value of the shipping group's `shippingAddress'` `companyName` property.
    - Iterate through shipping group names. If any name matches the shipping group's name, return the shipping group's `shippingGroupName`. Otherwise, return null.

- Create component based on `com.mystore.order.purchase.MyStoreShipToStoreShippingGroupInitializer`.
  - Name of the component is `/com/mystore/order/purchase/MyStoreShipToStoreShippingGroupInitializer`.
  - The component is request-scoped.
  - `storePropertyName` is the store.
  - `shippingGroupManager` is `/atg/commerce/order/shippingGroupManager`.
  - `shippingGroupType` is `shipToStoreShippingGroup`.
  - `shippingMethod` is `Ship to my home store`.

- Alter the `/atg/commerce/order/purchase/ShippingGroupDroplet` component.
  - Add `/com/mystore/order/purchase/MyStoreShipToStoreShippingGroupInitializer to its shippingGroupInitializers` property with a key of `shipToStoreShippingGroup`.

- Edit `/ae101/ATG10.1.1/MyStore/Storefront/j2ee-apps/Storefront/store.war/checkout/gadgets/shippingSingleForm.jsp`.
  - Add `shipToStoreShippingGroup` to the list of shipping group types for `ShippingGroupDroplet`.

- Testing:
  - Build, assemble, and deploy an EAR file to the production server.
  - Log in to the store as `lisa@example.com` (password is `password`). Add a product to the cart and follow the checkout process to the point of loading the shipping page.
  - Use `GenericSessionManager` to verify that the ship-to-store shipping group has been initialized in the `/atg/commerce/purchase/order/purchase/ShippingGroupContainerService`.

You are encouraged to try to complete this exercise using only the plan above. You should use the suggested variable, class, and component names.

Alternatively, there are step-by-step instructions starting on page 11 if you prefer.

Guided Instruction

Task 1: Creating a Custom Shipping Group Type

1. The `mys_store_ship_group` table has already been created for you in the `educommprod` and `educommpub` schemas of the Oracle database. View this table using `SQL*Plus`.
2. In Eclipse, switch to the Java EE perspective and create a new class called `ShipToStoreShippingGroup` in the `com.mystore.order` package. This class should extend `atg.commerce.order.HardgoodShippingGroup` and import `atg.repository.RepositoryItem`.
3. Add a `RepositoryItem` property named `store`, with the appropriate get and set methods.
4. Use a text editor or Eclipse to edit the `/atg/commerce/order/orderrepository.xml` file you created earlier under the **`MyStore.Custom`** `config` folder. Add the following lines just above the `</gsa-template>` tag:

```
<item-descriptor name="shippingGroup">
  <table name="dcspp_ship_group">
    <property name="type">
      <option value="shipToStoreShippingGroup"
       code="300"/>
    </property>
  </table>
</item-descriptor>

<item-descriptor name="shipToStoreShippingGroup" super-
type="hardgoodShippingGroup" sub-type-
value="shipToStoreShippingGroup">
```

(**Note:** In the preceding tag, there is no line break in the `super-type` and `sub-type-value` attributes.)

```
  <table name="mys_store_ship_group" type="auxiliary"
   id-column-name="ship_group_id">
    <property name="store" column-name="store_id"
     item-type="store"
     repository="/atg/store/stores/StoreRepository" />
  </table>

</item-descriptor>
```

5. To test your changes:
   - Open a command prompt and navigate to `<atgdir>/home/bin`.
   - Issue the following command (all on one line):

```
startSQLRepository -m MyStoreFakeXA -repository
/atg/commerce/order/OrderRepository
```

Resolve any errors before proceeding.

– Close the command window.

6. Use a text editor to edit the `/atg/commerce/order/OrderTools.properties` file you created under the **`MyStore.Custom`** `config` folder in an earlier lab. Add the following lines:

```
shippingTypeClassMap+=\
shipToStoreShippingGroup=\
com.mystore.order.ShipToStoreShippingGroup

shippingGroupShippingAddressClassNameMap+=\
shipToStoreShippingGroup=\
atg.commerce.order.RepositoryContactInfo
```

Also change the text that reads:

```
beanNameToItemDescriptorMap+=\
com.mystore.order.MyStoreOrder=order,\
com.mystore.pricing.MyStoreOrderPriceInfo=\
orderPriceInfo
```

to (additions in bold)

```
beanNameToItemDescriptorMap+=\
com.mystore.order.MyStoreOrder=order,\
com.mystore.pricing.MyStoreOrderPriceInfo=\
orderPriceInfo,\
com.mystore.order.ShipToStoreShippingGroup=\
shipToStoreShippingGroup
```

7. Rebuild project, assemble the `atg_production.ear` file, and deploy to WebLogic.
8. Use the Component Browser to verify that the properties have been set correctly in the `OrderTools` component, and that the `shipToStoreShippingGroup` item descriptor exists in the Order Repository with all the properties of a Hardgood Shipping Group, plus the `store` property.

Task 2: Creating a Custom Shipping Group Initializer

9. In Eclipse, create a new class named `MyStoreShipToStoreShippingGroupInitializer` in the `com.mystore.order.purchase` package. This class should extend `atg.nucleus.GenericService` and implement `atg.commerce.order.purchase.ShippingGroupInitializer` and `atg.commerce.order.purchase.ShippingGroupMatcher`.
10. Add the following import statements:

```
import atg.commerce.order.ShippingGroupManager;
import atg.repository.RepositoryItem;
import atg.beans.DynamicBeans;
```

```
import atg.commerce.order.RepositoryContactInfo;
import java.util.Set;
import java.util.Iterator;
import com.mystore.order.ShipToStoreShippingGroup;
```

11. Add the following properties, with the appropriate get and set methods:

```
private String storePropertyName;
private ShippingGroupManager shippingGroupManager;
private String shippingGroupType;
private String shippingMethod;
```

12. In the `initializeShippingGroups` method, change the default method signature to make the variable names more informative:

```
public void initializeShippingGroups(Profile
pProfile,ShippingGroupMapContainer pShippingGroupMapContainer,
DynamoHttpServletRequest pRequest)
```

13. Erase the TODO comment generated by Eclipse.
14. Add the following code:

```
String storePropertyName = getStorePropertyName();

if (storePropertyName == null)
 throw new ShippingGroupInitializationException
 ("No store property name set");

try {
  RepositoryItem store =
  (RepositoryItem)DynamicBeans.getSubPropertyValue
  (pProfile, storePropertyName);

  if (store == null) return;

    //create a shipping group
  ShipToStoreShippingGroup sg =
  (ShipToStoreShippingGroup)
  getShippingGroupManager().createShippingGroup
  (getShippingGroupType());

  RepositoryContactInfo sgAddress =
  (RepositoryContactInfo) sg.getShippingAddress();

  sg.setShippingMethod(getShippingMethod());
  sg.setStore(store);

  //copy address info from store item to sgAddress
  sgAddress.setAddress1((String)store.getPropertyValue
   ("address1"));
  sgAddress.setAddress2((String)store.getPropertyValue
   ("address2"));
  sgAddress.setAddress3((String)store.getPropertyValue
```

```
    ("address3"));
  sgAddress.setCity((String)store.getPropertyValue
    ("city"));
  sgAddress.setCounty((String)store.getPropertyValue
    ("county"));
  sgAddress.setCountry((String)store.getPropertyValue
    ("country"));
  sgAddress.setEmail((String)store.getPropertyValue
    ("email"));
  sgAddress.setFaxNumber((String)store.getPropertyValue
    ("faxNumber"));
  sgAddress.setPhoneNumber((String)store.getPropertyValue
    ("phoneNumber"));
  sgAddress.setPostalCode((String)store.getPropertyValue
    ("postalCode"));
  sgAddress.setState((String)store.getPropertyValue
    ("stateAddress"));
  sgAddress.setCompanyName((String)store.getPropertyValue
    ("storeName"));

  //copy name from profile
  sgAddress.setFirstName(
    (String)pProfile.getPropertyValue("firstName"));
  sgAddress.setMiddleName(
    (String)pProfile.getPropertyValue("middleName"));
  sgAddress.setLastName(
    (String)pProfile.getPropertyValue("lastName"));

    // add the ShippingGroup to ShippingGroupMapContainer:
  pShippingGroupMapContainer.addShippingGroup
    (sgAddress.getCompanyName(), sg);

} //end try

catch (Exception exc) {
  throw new ShippingGroupInitializationException(exc);
}
```

15. Change the signature of `getNewShippingGroupName()` to

```
public String getNewShippingGroupName(ShippingGroup pShippingGroup)
```

16. Erase the `TODO` comment and `return null;` and add the following code:

```
if (!pShippingGroup.getShippingGroupClassType().equals
("shipToStoreShippingGroup")) return null;

ShipToStoreShippingGroup sg =
 (ShipToStoreShippingGroup) pShippingGroup;

RepositoryContactInfo address =
 (RepositoryContactInfo) sg.getShippingAddress();
```

```
return address.getCompanyName();
```

17. Change the signature of `matchShippingGroup()` to:

```
public String matchShippingGroup(ShippingGroup pShippingGroup,
ShippingGroupMapContainer pShippingGroupMapContainer)
```

18. Erase the `TODO` comment and `return null;` and add the following code:

```
if (!pShippingGroup.getShippingGroupClassType().equals
("shipToStoreShippingGroup")) return null;

Set shippingGroupNames =
pShippingGroupMapContainer.getShippingGroupNames();

if (shippingGroupNames == null) return null;

Iterator nameIter = shippingGroupNames.iterator();
String shippingGroupName = null;
boolean found = false;

ShipToStoreShippingGroup sg =
 (ShipToStoreShippingGroup) pShippingGroup;

RepositoryContactInfo address =
 (RepositoryContactInfo) sg.getShippingAddress();

String thisShippingGroupName = address.getCompanyName();

while (nameIter.hasNext() && !found) {
  shippingGroupName = (String) nameIter.next();

  if (shippingGroupName.equals(thisShippingGroupName)) {
    found = true;
  } //end if
} //end while

if (found) return shippingGroupName;
else return null;
```

19. Build project and switch to the ATG perspective and the ATG Component Browser. Create a new component under `/com/mystore/order/purchase` named `MyStoreShipToStoreShippingGroupInitializer`. The component should be based on the `com.mystore.order.purchase.MyStoreShipToStoreShippingGroupInitializer` class and be request-scoped. Select the **Edit component after completion box**.
20. Set the new component's properties as follows:

    storePropertyName=`mystore`
    shippingGroupManager=`/atg/commerce/order/ShippingGroupManager`

shippingGroupType=`shipToStoreShippingGroup`
shippingMethod=`Ship to my home store`

21. Save the new component.
22. In the ATG Component Browser, open the `/atg/commerce/order/purchase/ShippingGroupDroplet` for editing. Right click the `shippingGroupInitializers` property and select edit. Select the last line and click **Add**. Enter `shipToStoreShippingGroup` as the key and `/com/mystore/order/purchase/MyStoreShipToStoreShippingGroupInitializer` as the value.
23. Save the changes to `ShippingGroupDroplet`.
24. Save a copy of `<atg_dir>/MyStore/Storefront/j2ee-apps/Storefront/store.war/checkout/gadgets/shippingSingleForm.jsp` to `<atg_dir>/MyStore/Storefront/j2ee-apps/Storefront/store.war/checkout/gadgets/shippingSingleForm.jsp.ORIG`.
25. Use a text editor to edit `<atg_dir>/MyStore/Storefront/j2ee-apps/Storefront/store.war/checkout/gadgets/shippingSingleForm.jsp`. Find the line that reads:

```
<dsp:param name="shippingGroupTypes"
value="hardgoodShippingGroup"/>
```

and change it to read:

```
<dsp:param name="shippingGroupTypes"
value="hardgoodShippingGroup,shipToStoreShippingGroup"/>
```

26. Rebuild the project, assemble the `atg_production.ear` file, and deploy to WebLogic.
27. In the store, log in as `lisa@example.com` (Lisa is the user you chose a home store for in an earlier lab; her password is `password`).
28. Add a product to Lisa's cart and click the **Checkout** link. On the cart page, click the **Checkout** button. This will take you to the shipping page, which will trigger the shipping groups to be initialized. You should see her selected store in the list of saved addresses.
29. In the browser admin interface, view Lisa's session. Set the context to **ATG Store** and navigate to `/atg/commerce/order/purchase/ShippingGroupContainerService`. Click the `shippingGroupMap` property. Confirm that her selected store is one of the shipping groups.

Practices for Lesson 16: Customizing Shipping Methods

Chapter 16

Practice 16-1: Customizing Shipping Methods

Overview

In the previous practice, you created a custom shipping type. In this practice, you will create a custom shipping method to use for that type. In this practice, you will create a custom shipping method which allows free shipping for in-store pickup. At the end of this exercise you will be able to:

- Create a custom shipping method
- Restrict when shipping methods are available
- Extend the validation of shipping groups

Problem Statement

Shipping to a store will always be free. You will implement a new ship-to-store shipping method that will only be available for ship-to-store shipping groups. The existing shipping methods will not be available for ship-to-store shipping groups.

You will need to add text to the default Java resource bundle, `atg.projects.store.web.WebAppResources`, to display to shoppers when the ship-to-store shipping method is offered.

The pipeline that validates shipping groups will need to be extended to validate ship-to-store shipping groups. The validator should ensure that the store still exists in the `StoreRepository`.

Planning

Take a moment before looking at the next page to think about what you will need to do to accomplish the tasks discussed above, as you have in the previous labs. A completed plan begins on the next page.

Plan

Task 1: Creating a Custom Shipping Method

- Create a class called `com.mystore.pricing.MyStoreShipToStoreShippingCalculator`.
  - Extends `atg.commerce.pricing.FixedPriceShippingCalculator`.
  - Add get and set methods for the `String` property `storeShippingGroupType`.
  - Implement `getAvailableMethods` method:
    - Test if type of `pShippingGroup` matches `storeShippingGroupType`.
    - If so, call `super.getAvailableMethods()`.

- Create a component based on `com.mystore.pricing.MyStoreShipToStoreShippingCalculator`.
  - Name of the component is `/com/mystore/pricing/shipping/MyStoreShipToStoreShippingCalculator`.
  - `storeShippingGroupType` is `shipToStoreShippingGroup`.
  - `amount` is `0`.
  - `pricingTools` is `/atg/commerce/pricing/PricingTools`.
  - `shippingMethod` is `Ship to my home store`.

- Alter the `/atg/commerce/pricing/ShippingPricingEngine` component.
  - Add `/com/mystore/pricing/shipping/MyStoreShipToStoreShippingCalculator` to its `preCalculators` property.

- Create a class called `com.mystore.pricing.MyStoreFixedPriceShippingCalculator`.
  - Extend `atg.commerce.pricing.FixedPriceShippingCalculator`.
  - Add get and set methods for String `storeShippingGroupType` property.
  - Implement `getAvailableMethods` method:
    - Test if the type of `pShippingGroup` matches `storeShippingGroupType`.
    - If not, call `super.getAvailableMethods()`.

- Create a class called `com.mystore.pricing.MyStorePriceRangeShippingCalculator`.
  - Extend `atg.commerce.pricing.PriceRangeShippingCalculator`.
  - Add get and set methods for String `storeShippingGroupType` property.
  - Implement the `getAvailableMethods` method:

        - Test if the type of `pShippingGroup` matches `storeShippingGroupType`.
        - If not, call `super.getAvailableMethods()`.

- Alter the `/atg/commerce/pricing/shipping/NextDay` component.
    - Change class to `com.mystore.pricing.MyStoreFixedPriceShippingCalculator`.
    - Set `storeShippingGroupType` to `shipToStoreShippingGroup`.

- Alter the `/atg/commerce/pricing/shipping/TwoDay` component.
    - Change class to `com.mystore.pricing.MyStoreFixedPriceShippingCalculator`.
    - Set `storeShippingGroupType` to `shipToStoreShippingGroup`.

- Alter the `/atg/commerce/pricing/shipping/Ground` component.
    - Change class to `com.mystore.pricing.MyStorePriceRangeShippingCalculator`.
    - Set `storeShippingGroupType` to `shipToStoreShippingGroup`.

- Copy the `atg.projects.store.web.WebAppResources.properties` file from **EStore** `classes.jar` to the **`MyStore.Custom`** `src` folder.
    - Add custom user message to file.
        - Key is `checkout_shipping.deliveryShiptomyhomestore`, value is `Ship to my home store`.

- Testing:
    - Build, assemble, and deploy an EAR file to the production server.
    - Log in to the **ATG Store** as `lisa@example.com` (password is `password`). Add a product to the cart and follow the checkout process to the point of loading the shipping page.
    - On the shipping page, select to ship to the store and proceed to the shipping methods page.
    - Confirm that only the `Ship to my home store` method is offered.
    - Use the Back button to return to the shipping page and select to ship to Lisa's home address.
    - Confirm that the available shipping methods do not include `Ship to my home store`.

Task 2: Extending the Shipping Group Verification Pipelines

- Create a class called `com.mystore.order.processor.MyStoreProcValidateShipToStoreShippingGroup`.
    - Extend `atg.nucleus.logging.ApplicationLoggingImpl`.

    - Implement `atg.service.pipeline.PipelineProcessor`.
    - Implement the `runProcess` method:
        - Get the shipping group from `pParam`, cast as `ValidateShippingGroupPipelineArgs`.
        - If the shipping group is null or not an instance of `ShipToStoreShippingGroup`, throw exception.
        - Cast the shipping group as `ShipToStoreShippingGroup`.
        - Get store item from `store` property of shipping group.
        - Include a debugging message with the shipping group ID.
        - If the store item is null, add error to `pResult`.

- Create a component based on `com.mystore.order.processor.MyStoreProcValidateShipToStoreShippingGroup`.
    - The name of the component is `/com/mystore/order/processor/ValidateShipToStoreShippingGroup`.
    - `storeRepository` is `/atg/store/stores/StoreRepository`.
    - `loggingDebug` is `true`.
    - `loggingIdentifier` is `MyStoreProcValidateShipToStoreShippingGroup`.

- Alter the `/atg/commerce/order/processor/ValidateShippingGroupByType` component.
    - Add `shipToStoreShippingGroup=5000` to its `returnValues` property.

- Extend `/atg/commerce/commercepipeline.xml` under **`MyStore.Custom`**.
    - Extend pipeline chain `validateShippingGroup`.
        - Extend pipeline link `dispatchOnSGType`.
            - Add transition
                - `returnvalue is 5000`
                - `link is validateShipToStoreSG`
        - Add a pipeline link named `validateShipToStoreSG`.
            - Transaction is `TX_MANDATORY`
            - Processor is `/com/mystore/order/processor/ValidateShipToStoreShippingGroup`

- Testing:
    - Build, assemble, and deploy an EAR file to the production server.
    - Use the **Dynamo Admin UI** Component Browser to confirm that changes have been made to `commercepipeline.xml` (viewable through `/atg/commerce/PipelineManager`'s `definitionFile` property).

- Return to the **ATG Store** and restart the checkout process as `lisa@example.com`. Follow the checkout process to the point of loading the shipping page.
- On the shipping page, select to ship to the store and proceed to the shipping methods page.
- Check the production server console for the debugging message from `MyStoreProcValidateShipToStoreShippingGroup`.

You are encouraged to try to complete this exercise using only the plan above. You should use the suggested variable, class, and component names.

Alternatively, there are step-by-step instructions starting on the next page if you prefer.

Guided Instruction

Task 1: Creating a Custom Shipping Method

1. In Eclipse, switch to the Java EE perspective and create a new class named `MyStoreShipToStoreShippingCalculator` in the `com.mystore.pricing` package. This class should extend `atg.commerce.pricing.FixedPriceShippingCalculator`.
2. Add the following import statements:

```
import atg.commerce.order.ShippingGroup;
import java.util.List;
import atg.repository.RepositoryItem;
import java.util.Locale;
import java.util.Map;
import atg.commerce.pricing.PricingException;
```

3. Add a String property named `storeShippingGroupType`, with the appropriate get and set methods.
4. Override the `getAvailableMethods` method:

```
public void getAvailableMethods(List pMethods, ShippingGroup
pShippingGroup, RepositoryItem pPricingModel, Locale pLocale,
RepositoryItem pProfile, Map pExtraParameters) throws
PricingException {

  if (pShippingGroup.getShippingGroupClassType().equals
   (getStoreShippingGroupType())) {

    super.getAvailableMethods(pMethods, pShippingGroup,
     pPricingModel, pLocale, pProfile, pExtraParameters);
  } //end if
```

5. Build project and switch to the ATG perspective and the ATG Component Browser. Create a new component under `/com/mystore/pricing/shipping` named `MyStoreShipToStoreShippingCalculator`. The component should be based on the `com.mystore.pricing.MyStoreShipToStoreShippingCalculator` class. Select the **Edit component after completion** box.
6. Set the new component's properties as follows:

   storeShippingGroupType=`shipToStoreShippingGroup`
   amount=`0`
   pricingTools=`/atg/commerce/pricing/PricingTools`
   shippingMethod=`Ship to my home store`
7. Save the new component.
8. Using a text editor, create a `/atg/commerce/pricing/ShippingPricingEngine.properties` file under the **`MyStore.Custom`** `config` folder. Add the following lines:

```
preCalculators+=\
/com/mystore/pricing/shipping/MyStoreShipToStoreShippingCalculator
```
(**Note:** The component path should be all on one line.)

9. Copy the `MyStoreFixedPriceShippingCalculator.java` and `MyStorePriceRangeShippingCalculator.java` files from the `<coursedir>/src/Lesson16` directory to `MyStore/Custom/src/com/mystore/pricing`. Refresh the view in Eclipse and open these files.
10. Review the contents of these files. In particular, the `getAvailableMethods` method has been overridden so that these calculators will not add themselves to the list if the shipping group is ship-to-store.
11. Save both files and build project. You do not need to make any changes to either file.
12. Using a text editor or Eclipse, create a `/atg/commerce/pricing/shipping/NextDay.properties` file under the **`MyStore.Custom`** `config` folder. Add the following lines:

```
$class=\
com.mystore.pricing.MyStoreFixedPriceShippingCalculator
storeShippingGroupType=shipToStoreShippingGroup
```

13. Using a text editor or Eclipse, create a `/atg/commerce/pricing/shipping/TwoDay.properties` file under the **`MyStore.Custom`** `config` folder. Add the following lines:

```
$class=\
com.mystore.pricing.MyStoreFixedPriceShippingCalculator
storeShippingGroupType=shipToStoreShippingGroup
```

14. Using a text editor or Eclipse, create a `/atg/commerce/pricing/shipping/Ground.properties` file under the **`MyStore.Custom`** `config` folder. Add the following lines:

```
$class=\
com.mystore.pricing.MyStorePriceRangeShippingCalculator
storeShippingGroupType=shipToStoreShippingGroup
```

15. In Eclipse, switch to the Java EE perspective. Expand `Commerce Reference Store's Store/Storefront/lib` **`classes.jar`** (which should be near the end), then **`atg.projects.store.web`**. Open the **`WebAppResources.properties`** file.
16. Save a copy of this file in your own module by selecting **File > Save as**. In the **Enter or select parent folder** field, enter `MyStore.Custom/src/atg/projects/store/web`. In the **File name** field, enter `WebAppResources.properties`. Click **OK**.
17. Your new copy of **`WebAppResources.properties`** will now be open for editing. Add the following lines to the bottom of this file:

```
#MyStore custom messages
common.deliveryShiptomyhomestore=Ship to my home store
checkout_shipping.deliveryShiptomyhomestore=Ship to my home store
```

Note:

- The display text should be all on one line.
- For a multilingual site, you would repeat this step for the resource bundles tied to other locales.

18. Save the file.
19. Rebuild project, assemble the `atg_production.ear` file, and deploy to WebLogic.
20. Open the home page of the store. If you use the same browser as your previous test session, you will still be logged in as Lisa. If not, log in as Lisa. Click the **Checkout** link to restart the checkout process.
21. Click the **Checkout** button on the cart page. If you were previously logged in, you will be asked to provide your password again.
22. On the shipping page, select your store and click the **Ship to this Address** button. On the next page, the **Ship to my home store** shipping method should be the only option, with a price of $0.
23. Click the browser's Back button to return to the shipping page. Select Lisa's home address and click the **Ship to this Address** button. You should now see the other shipping methods, but not the ship-to-store method.

Task 2: Extending the Shipping Group Verification Pipelines

24. In Eclipse, switch to the Java EE perspective and create a new class named `MyStoreProcValidateShipToStoreShippingGroup` in the `com.mystore.order.processor` package. This class should extend `atg.nucleus.logging.ApplicationLoggingImpl` and implement `atg.service.pipeline.PipelineProcessor`.
25. Add the following import statements:

```
import java.util.ResourceBundle;
import atg.core.i18n.LayeredResourceBundle;
import atg.commerce.order.InvalidParameterException;
import atg.commerce.order.ShippingGroup;
import atg.core.util.ResourceUtils;
import
atg.commerce.order.processor.ValidateShippingGroupPipelineArgs;
import com.mystore.order.ShipToStoreShippingGroup;
import atg.repository.RepositoryItem;
```

26. Add the following code to access the standard resource bundle:

```
static final String RESOURCE_NAME =
  "atg.commerce.order.OrderResources";

private static ResourceBundle sResourceBundle =
  LayeredResourceBundle.getBundle(RESOURCE_NAME,
  atg.service.dynamo.LangLicense.getLicensedDefault());
```

27. Add the following code above the `getRetCodes` method:

```
private final int SUCCESS = 1;
protected int[] mRetCodes = { SUCCESS };
```

28. In the `getRetCodes` method, erase the `TODO` comment and replace `return null;` with `return mRetCodes;`.
29. In the `runProcess` method, change the method arguments from `arg0` and `arg1` to `pParam` and `pResult`.
30. In the body of the `runProcess` method, erase the `TODO` comment and `return 0;` and add the following code:

```
ValidateShippingGroupPipelineArgs args =
(ValidateShippingGroupPipelineArgs)pParam;

ShippingGroup shippingGroup = args.getShippingGroup();

if (shippingGroup == null)
  throw new InvalidParameterException
   (ResourceUtils.getMsgResource
   ("InvalidShippingGroupParameter", RESOURCE_NAME,
    sResourceBundle));

if (!(shippingGroup instanceof ShipToStoreShippingGroup))
  throw new InvalidParameterException
   (ResourceUtils.getMsgResource
   ("InvalidShippingGroupParameter", RESOURCE_NAME,
    sResourceBundle));

ShipToStoreShippingGroup storeShippingGroup =
(ShipToStoreShippingGroup) shippingGroup;

RepositoryItem store = storeShippingGroup.getStore();

if (isLoggingDebug())
  logDebug("Validating store shipping group " +
   shippingGroup.getId());

if (store == null) {
  pResult.addError("StoreNoLongerExists",
   "The store selected for shipping no longer exists.");
}
return SUCCESS;
```

31. In the Package Explorer tab, create a new component under `/com/mystore/order/processor` named `ValidateShipToStoreShippingGroup`.
32. Add the following lines to the properties file and save the component:

```
$class=com.mystore.order.processor.MyStoreProcValidateShipToStoreSh
ippingGroup
storeRepository=/atg/store/stores/StoreRepository
```

```
loggingDebug=true
loggingIdentifier=MyStoreProcValidateShipToStoreShippingGroup
```

33. Open the `/atg/commerce/order/processor/ValidateShippingGroupByType` component for editing. Right click the `returnValues` property and select edit. Select the last line and click **Add**. Enter `shipToStoreShippingGroup` as the key and `5000` as the value.
34. Click **OK** to save property.
35. Save the component.
36. Using a text editor or Eclipse, create a file named `/atg/commerce/commercepipeline.xml` under the **`MyStore.Custom`** `config` folder.
37. Add the following content to this file and save it:

```
<pipelinemanager>
  <pipelinechain name="validateShippingGroup">
    <pipelinelink name="dispatchOnSGType">
      <transition returnvalue="5000"
       link="validateShipToStoreSG"/>
    </pipelinelink>

    <pipelinelink name="validateShipToStoreSG"
     transaction="TX_MANDATORY">
      <processor jndi=
       "/com/mystore/order/processor/
        ValidateShipToStoreShippingGroup" />
```

(**Note:** The `jndi` attribute above should be all on one line)

```
  </pipelinelink>
  </pipelinechain>
</pipelinemanager>
```

38. Rebuild project, assemble the `atg_production.ear` file, and deploy to WebLogic.
39. In the Component Browser, browse to `/atg/commerce/PipelineManager`. Click the link for its `definitionFile` property to view the combined version of `commercepipeline.xml`. Locate the `validateShippingGroup` pipeline and confirm that your changes have been included.
40. Navigate to the `/atg/commerce/order/processor/ValidateShippingGroupByType` component and confirm that your changes have been made to the `returnValues` property.

41. In the store, click the **Checkout** link to begin the checkout process again. On the shipping page, select your store and click the **Ship to this Address** button.
42. Switch to the Production server console. You should see a debugging message from `MyStoreProcValidateShipToStoreShippingGroup` similar to:

```
**** debug      Thu Dec 20 14:31:49 EST 2012    1356031909598
MyStoreProcValidateShipToStoreShippingGroup     Validating store
shipping group sg260009
```

Practices for Lesson 17: Extending Payment Groups

Chapter 17

Practice 17-1: Extending Payment Groups

Overview

In this practice, you will be extending payment groups which will allow shoppers to purchase using their points. At the end of this exercise you will be able to:

- Create a custom payment method
- Create a custom payment method initializer
- Create a pipeline processor
- Create a new pipeline
- Extend an existing pipeline

Problem Statement

In order to allow shoppers to pay for reward items with points, you need to implement a custom payment method.

As part of this, you will be extending the Profile Repository to create a new points object and modifying the user profile's points property to use to this object instead of the simple integer property you created earlier. The points object will have properties for the available points balance and the reserved points balance (used to reserve points for pending orders). The user's points object should be created, updated, and deleted along with their profile.

Because you are basing your implementation on CRS, you will need to extend the `BillingInfoFormHandler` and `BillingProcessHelper` to initialize the points payment groups and allocate the correct amount of the order to them.

You will extend the pipeline that validates payment groups to include a new processor that validates points payment groups, and create a new pipeline to authorize, credit, and debit these groups. You will also extend the pipeline that handles credit card debits to award points after regular orders have been paid for.

Planning

Take a moment before looking at the next page to think about what you will need to do to accomplish the tasks discussed above, as you have in the previous labs. A completed plan begins on the next page.

Plan

Task 1: Extending the Profile Repository

- Create a table called `mys_points` in production and publishing datasources.
  - The new table will have three columns: `points_id` (`varchar2(40)`), `available` (`number(10)`), and `reserved` (`number(10)`).

- Create table called `mys_user_points` in production and publishing datasources.
  - The new table will have two columns: `user_id` (`varchar2(40)`) and `points_id` (`varchar2(40)`).

- Extend `userProfile.xml` under **`MyStore.Custom`**.
  - Create new item descriptor named `points`.
    - Table is `mys_points`.
    - Table is `primary`.
    - `availableBalance` is `int` property linked to `available` column.
    - `reservedBalance` is `int` property linked to `reserved` column.
  - Extend item descriptor `user`.
    - Delete definition of previous `points` property.
    - Add `mys_user_points` table.
      - Table is `auxiliary`.
      - `points` is property of item type `points` linked to `points_id` column.
        - Specify `cascade="insert,update,delete"`.

- Alter the configuration of `/com/mystore/points/AddPointsActionConfig`.
  - Change `pointsPropertyName` from `points` to `points.availableBalance`.

- Testing:
  - Use `startSQLRepository` to test extensions to `ProfileAdapterRepository`.
  - Build, assemble, and deploy an EAR file to the production server.
  - Open the **ACC** and list the users. Create points object with balance of `700` for `eric@example.com`.

Task 2: Creating a New Payment Group Type

- Create a table called `mys_points_pg` in production and publishing datasources.
  - The new table will have two columns: `payment_group_id (varchar2(40))` and `points_item_id (varchar2(40))`.

- Extend `orderrepository.xml` under **`MyStore.Custom`**.
  - Extend item descriptor `paymentGroup`.
  - Property to extend is `type`.
  - Add `option` tag with the value `pointsPaymentGroup` and code `300`.
  - Create new item descriptor named `pointsPaymentGroup`.
    - `super-type` is `paymentGroup`.
    - Table is `mys_ points_pg`.
    - Table is auxiliary.
    - Property is `pointsItem`, pointing to `points` item type in `/atg/userprofiling/ProfileAdapterRepository`.

- Create a class called `com.mystore.order.PointsPaymentGroup`.
  - Extend `atg.commerce.order.PaymentGroupImpl`.
  - Add get and set methods for RepositoryItem `pointsItem` property.
    - Methods should use `getPropertyValue()` and `setPropertyValue()`.

- Alter properties of `OrderTools`.
  - Add `com.mystore.order.PointsPaymentGroup` to `beanNameToItemDescriptorMap` property, with key `pointsPaymentGroup`.
  - Add `com.mystore.order.PointsPaymentGroup` to property `paymentTypeClassMap`, with key `pointsPaymentGroup`.

- Testing:
  - Use `startSQLRepository` to test extensions to `OrderRepository`.
  - Build, assemble, and deploy an EAR file to the production server.
  - Verify that `OrderTools` properties are set correctly.
  - Verify that `pointsPaymentGroup` item descriptor exists in the `OrderRepository`.

Task 3: Adding a Payment Group Validator

- Create a class called `com.mystore.order.processor.MyStoreProcValidatePointsPaymentGroup`
  - Extend `atg.nucleus.logging.ApplicationLoggingImpl`.
  - Implement `atg.service.pipeline.PipelineProcessor`.
  - Implement `runProcess` method:
    - Get payment group and order from `pParam`, cast as `ValidatePaymentGroupPipelineArgs`.
    - If payment group is null or not an instance of `PointsPaymentGroup`, throw exception.
    - Cast `order` as `MyStoreOrder`.
    - If `order` is not a reward `order`, add error to `pResult` and return.
    - Cast payment group as `PointsPaymentGroup`.
    - If payment group's `pointsItem` is null, add error to `pResult`.

- Create a component based on `com.mystore.order.processor.MyStoreProcValidatePointsPaymentGroup`.
  - Name of the component is `/com/mystore/order/processor/ValidatePointsPaymentGroup`.
  - `loggingDebug` is `true`.
  - `loggingIdentifier` is `MyStoreProcValidatePointsPaymentGroup`.

- Alter the `/atg/commerce/order/processor/ValidatePaymentGroupByType` component.
  - Add `pointsPaymentGroup=6000` to its `returnValues` property.

- Extend `/atg/commerce/commercepipeline.xml` under **`MyStore.Custom`**.
  - Extend the pipeline chain `validatePaymentGroup`.
    - Extend the pipeline link `dispatchOnPGType`.
      - Add transition
        - `returnvalue` is `6000`
        - `link` is `validatePointsPG`
    - Add pipeline link named `validatePointsPG`.
      - Transaction is `TX_MANDATORY`
      - Processor is `/com/mystore/order/processor/ValidatePointsPaymentGroup`
  - Extend the pipeline chain `validatePaymentGroupPreConfirmation`.
    - Extend the pipeline link `dispatchOnPGTypePreConfirmation`.
      - Add transition
        - `returnvalue` is `6000`

        - link is validatePointsPGPreConfirmation
    - Add a pipeline link named validatePointsPGPreConfirmation.
        - Transaction is TX_MANDATORY
        - Processor is /com/mystore/order/processor/ValidatePointsPaymentGroup

- Testing:
    - Build, assemble, and deploy an EAR file to the production server.
    - Use the **Dynamo Admin UI** Component Browser to confirm that changes have been made to commercepipeline.xml (viewable through the /atg/commerce/PipelineManager definitionFile property).

Task 4: Adding Points Functionality to BillingProcessHelper and BillingInfoFormHandler

- Create a class called com.mystore.order.purchase.MyStoreBillingProcessHelper.
    - Extend atg.projects.store.order.purchase.StoreBillingProcessHelper.
    - Implement the initializePointsPaymentGroup method.
        - Test if order is a reward order. If yes, get points item from profile.
        - If points item is null, throw exception.
        - If points item is not null, instantiate new PointsPaymentGroup.
        - Check the order's paymentGroups for existing points payment group.
        - If there is an existing group, use this group. If not, use PaymentGroupManager to create a new one and add it to the order.
        - Loop through commerce items in order. Add their amounts together to determine points needed.
        - If points needed for order are more than the point item's availableBalance, throw exception.
        - Use PaymentGroupManager to clear all payment group relationships from order.
        - Use OrderManager to assign points needed to points payment group.

- Create a class called com.mystore.order.purchase.MyStoreBillingInfoFormHandler.
    - Extend atg.projects.store.order.purchase.BillingInfoFormHandler.
    - Implement the preSetupStoreCreditPaymentGroupsForOrder method.
        - Get the billingHelper property, cast as MyStoreBillingHelper.
        - Get the order property, cast as MyStoreOrder.

        - If `order` is reward order, call `helper.initializerPointsPaymentGroup`.

- Alter the `/atg/store/order/purchase/BillingProcessHelper` component.
    - Change class to `com.mystore.order.purchase.MyStoreBillingProcessHelper`.

- Alter the `/atg/store/order/purchase/BillingFormHandler` component.
    - Change class to `com.mystore.order.purchase.MyStoreBillingInfoFormHandler`.

- Testing:
    - Build, assemble, and deploy an EAR file to the production server.
    - Log in to the **US Reward Site** as `eric@example.com` (password is `password`).
    - Add a Paris Clock to the cart and follow the checkout process to the point of loading the billing page.
    - Use `GenericSessionManager` to confirm that the order's `paymentGroups` includes a `pointsPaymentGroup`, and that the order's `paymentGroupRelationships` are allocated correctly.

Task 5: Adding a Payment Group Payment Processor

- Create a class called `com.mystore.payment.processor.MyStoreProcProcessPointsPayment`.
  - Extend `atg.commerce.payment.processor.ProcProcessPaymentGroup`.
  - Add `MutableRepository` property named `profileRepository`.
  - Implement the `authorizePaymentGroup` method.
    - Get `amount` and `order` from `pParams`.
    - Get `profileId` from order.
    - Get the profile's `points` item from `profileRepository` as a `MutableRepositoryItem`.
    - Get values of `reservedBalance` and `availableBalance`.
    - Check that `amount` is not greater than `availableBalance`; if it is, return new `PaymentStatusImpl` that indicates payment failed.
    - Subtract amount from `availableBalance`.
    - Add amount to `reservedBalance`.
    - Set new values for `reservedBalance` and `availableBalance` in `points` item.
    - Update `points` item.
    - Return new `PaymentStatusImpl` that indicates that the payment succeeded.
  - Implement the `creditPaymentGroup` method.
    - Get `amount` and `order` from `pParams`.
    - Get `profileId` from order.
    - Get profile's `points` item from `profileRepository` as `MutableRepositoryItem`.
    - Get values of `reservedBalance` and `availableBalance`.
    - Subtract `amount` from `reservedBalance`.
    - Add `amount` to `availableBalance`.
    - Set new values for `reservedBalance` and `availableBalance` in `points` item.
    - Update `points` item.
    - Return new `PaymentStatusImpl` that indicates payment succeeded.
  - Implement `debitPaymentGroup` method.
    - Get `amount` and order from `pParams`.
    - Get `profileId` from order.
    - Get profile's `points` item from `profileRepository` as `MutableRepositoryItem`.
    - Get values of `reservedBalance` and `availableBalance`.
    - Subtract `amount` from `reservedBalance`.

        - Set new value for `reservedBalance` in `points` item.
        - Update `points` item.
        - Return new `PaymentStatusImpl` that indicates payment succeeded.

- Create component based on `com.mystore.payment.processor.MyStoreProcProcessPointsPayment`.
    - Name of the component is `/com/mystore/payment/processor/ProcessPointsPayment`.
    - `profileRepository` is `/atg/userprofiling/ProfileAdapterRepository`.

- Extend `/atg/commerce/payment/paymentpipeline.xml` under **`MyStore.Custom`**.
    - Create a pipeline chain named `pointsPaymentProcessorChain`.
        - `transaction` is `TX_REQUIRED`.
        - `headlink` is `processPointsPayment`.
        - `pipelinelink` is named `processPointsPayment`.
            - Transaction is `TX_MANDATORY`.
            - Processor is `/com/mystore/payment/processor/ProcessPointsPayment`.

- Alter the `/atg/commerce/payment/PaymentManager` component.
    - Add `com.mystore.order.PointsPaymentGroup` to its `paymentGroupToChainNameMap` property with a key of `pointsPaymentProcessorChain`.

- Testing:
    - Build, assemble, and deploy an EAR file to the production server.
    - Use the **Dynamo Admin UI** Component Browser to verify that `pointsPaymentProcessorChain` is on `/atg/commerce/payment/PaymentPipelineManager`'s list of pipelines.
    - Verify that changes have been made to `/atg/commerce/payment/PaymentManager.paymentGroupToChainNameMap`.
    - Return to the **US Reward Site**. Restart checkout process as `eric@example.com`. Complete order (T**ip**: Enter `111` for CSV code for stored credit card). Note the order ID on the confirmation page.
    - Open Production **ACC**. Look up users and confirm that Eric's points item has been modified correctly.
    - In the **Dynamo Admin UI**, use the **Commerce Administration > Fulfillment Administration** section to report that Eric's order has shipped.
    - Return to the **ACC** and re-open Eric's `points` item. Confirm that the `reservedBalance` is now `0`.

Task 6: Extending Payment Processing to Award Points for Regular Orders

- Create a class called `com.mystore.payment.processor.MyStoreProcAwardPoints`.
  - Extend `atg.nucleus.GenericService`.
  - Implement `atg.service.pipeline.PipelineProcessor`.
  - Add a `MutableRepository` property named `profileRepository`.
  - Implement the `runProcess` method:
    - Cast `pParams` as `PaymentManagerPipelineArgs`.
    - Get `paymentStatus`, order, and action from `pParams`.
    - If `paymentStatus' transactionSuccess` is `true`, and `action` is `PaymentManagerAction.DEBIT`, and order is not a reward order:
      - Get `points` amount from order.
      - Get `points` item for order's profile.
      - Add `points` amount from order to the `point`'s `availableBalance`.
      - Update points item.

- Create a component based on `com.mystore.payment.processor.MyStoreProcAwardPoints`.
  - Name of the component is `/com/mystore/payment/processor/AwardPoints`.
  - `profileRepository` is `/atg/userprofiling/ProfileAdapterRepository`.

- Extend `/atg/commerce/payment/paymentpipeline.xml` under **`MyStore.Custom`**.
  - Extend the pipeline chain `creditCardProcessorChain`.
    - Extend the pipeline link `processCreditCard`.
      - Add transition link to `awardPoints` when `returnvalue` is `1`.
    - Add pipeline link `awardPoints`.
      - Transaction is `TX_MANDATORY`.
      - Processor is `/com/mystore/payment/processor/AwardPoints`.
- Testing:
  - Build, assemble, and deploy an EAR file to the production server.
  - In the **ATG Store**, log in as `eric@example.com` and place an order.
  - In the **Dynamo Admin UI**, use **Commerce Administration > Fulfillment Administration** section to report that Eric's order has shipped.
  - Open the **ACC** and confirm that Eric has been awarded correct number of points.

You are encouraged to try to complete this exercise using only the plan above. You should use the suggested variable, class, and component names.

Alternatively, there are step-by-step instructions starting on the next page if you prefer.

Guided Instruction

Task 1: Extending the Profile Repository

1. The `mys_points` and `mys_user_points` tables have already been created for you in the `educommprod` and `educommpub` schemas of the Oracle database. View this table using `SQL*Plus`.
2. Using a text editor or Eclipse, edit the `/atg/userprofiling/userProfile.xml` file under the **`MyStore.Custom`** `config` folder. Add the following lines at the bottom, just above `</gsa-template>`:

```
<item-descriptor name="points">
 <table name="mys_points" type="primary"
  id-column-name="points_id">
   <property name="availableBalance"
    column-name="available" data-type="int"/>
   <property name="reservedBalance"
    column-name="reserved" data-type="int"/>
 </table>
</item-descriptor>
```

3. Find the `points` property you configured earlier and delete that line:

```
<property name="points" display-name="Loyalty Points" category="My
Store Extensions" data-type="int" column-name="points" default="0"
/>
```

4. In the user item descriptor, add the following lines, just above the `</item-descriptor>` tag:

```
<table name="mys_user_points" type="auxiliary"
 id-column-name="user_id">
   <property name="points" column-name="points_id"
    item-type="points" display-name="Loyalty Points"
    category="My Store Extensions"
    cascade="insert,update,delete"/>
</table>
```

5. Save `userProfile.xml`.
6. To test your changes:
   - Open a command prompt and navigate to `<atgdir>\home\bin`.
   - Issue the following command (all on one line):

     ```
     startSQLRepository -m MyStoreFakeXA -repository
     /atg/userprofiling/ProfileAdapterRepository
     ```

     Resolve any errors before proceeding.
   - Leave the command window open for later testing.

7. In Eclipse, switch to the ATG perspective and the Component Browser. Open the `/com/mystore/points/AddPointsActionConfig` component for editing.
8. Change the value of `pointsPropertyName` from `points` to `points.availableBalance` and save the component.
9. Rebuild project, assemble the `atg_production.ear` file, and deploy to WebLogic.
10. Open the **ACC** and navigate to **People & Organizations > Users**. Click the **List** button to view the users.
11. Select **eric@example.com** and find his **Loyalty Points** property. Click this field, then click the **…** button.
12. In the pop up window, click the **New Item** button.
13. In the next pop up window, enter `700` in the `availableBalance` field, and click **OK**. Navigate to **File > Save** to save changes to eric's profile.

Task 2: Creating a New Payment Group Type

14. The `mys_poings_pg` table has already been created for you in the `educommprod` and `educommpub` schemas of the Oracle database. View this table using `SQL*Plus`.
15. In a text editor or Eclipse, open the `/atg/commerce/order/orderrepository.xml` file you created in an earlier lesson. Add the following lines just above the `</gsa-template>` tag:

```
<item-descriptor name="paymentGroup">
  <table name="dcspp_pay_group">
    <property name="type">
      <option value="pointsPaymentGroup" code="300"/>
    </property>
  </table>
</item-descriptor>

<item-descriptor name="pointsPaymentGroup"
  super-type="paymentGroup"
  sub-type-value="pointsPaymentGroup">

  <table name="mys_points_pg" type="auxiliary"
   id-column-name="payment_group_id">

    <property name="pointsItem"
      column-name="points_item_id" item-type="points"
      repository=
      "/atg/userprofiling/ProfileAdapterRepository"/>
  </table>
</item-descriptor>
```

16. To test your changes:
    - Open a command prompt and navigate to `<atgdir>\home\bin`.
    - Issue the following command (all on one line):

```
startSQLRepository -m MyStoreFakeXA -repository
/atg/commerce/order/OrderRepository
```

Resolve any errors before proceeding.

- Close the command window.

17. In Eclipse, switch to the Java EE perspective and create a class named `PointsPaymentGroup` in the `com.mystore.order` package. This class should extend `atg.commerce.order.PaymentGroupImpl`.
18. Add the following import statement:

```
import atg.repository.RepositoryItem;
```

19. Add a `RepositoryItem` property called `pointsItem`.
20. Implement the `getPointsItem` method:

```
public RepositoryItem getPointsItem() {
  return (RepositoryItem) getPropertyValue("pointsItem");
}
```

21. Implement the `setPointsItem` method:

```
public void setPointsItem(RepositoryItem pointsItem) {
  setPropertyValue("pointsItem", pointsItem);
}
```

22. Save the class. Watch for Java errors.
23. Use a text editor to edit `/atg/commerce/order/OrderTools.properties` under the **`MyStore.Custom`** `config` folder created in an earlier lesson. Add the following lines:

```
paymentTypeClassMap+=\
pointsPaymentGroup=\
com.mystore.order.PointsPaymentGroup
```

Also change the text that reads:

```
beanNameToItemDescriptorMap+=\
com.mystore.order.MyStoreOrder=order,\
com.mystore.pricing.MyStoreOrderPriceInfo=\
orderPriceInfo,\
com.mystore.order.ShipToStoreShippingGroup=\
shipToStoreShippingGroup
```

to (additions in bold)

```
beanNameToItemDescriptorMap+=\
com.mystore.order.MyStoreOrder=order,\
com.mystore.pricing.MyStoreOrderPriceInfo=\
orderPriceInfo,\
com.mystore.order.ShipToStoreShippingGroup=\
shipToStoreShippingGroup,\
com.mystore.order.PointsPaymentGroup=\
pointsPaymentGroup
```

24. Rebuild project, assemble the `atg_production.ear` file, and deploy to WebLogic.
25. Use the **Dynamo Admin UI** Component Browser to confirm that the `OrderTools` component has the correct property values.

Task 3: Adding a Payment Group Validator

26. In Eclipse, switch to the Java EE perspective and create a new class named `MyStoreProcValidatePointsPaymentGroup` in the `com.mystore.order.processor` package. This class should extend `atg.nucleus.logging.ApplicationLoggingImpl` and implement `atg.service.pipeline.PipelineProcessor`.
27. Add the following import statements:

```
import java.util.ResourceBundle;
import atg.core.i18n.LayeredResourceBundle;
import atg.commerce.order.InvalidParameterException;
import com.mystore.order.MyStoreOrder;
import com.mystore.order.PointsPaymentGroup;
import atg.commerce.order.Order;
import atg.commerce.order.PaymentGroup;
import atg.commerce.order.processor.
  ValidatePaymentGroupPipelineArgs;
import atg.core.util.ResourceUtils;
import atg.repository.RepositoryItem;
```

(**Note:** The import statement for `ValidatePaymentGroupPipelineArgs` should be all on one line.)28. Add the following code to access the standard resource bundle:

```
static final String RESOURCE_NAME =
  "atg.commerce.order.OrderResources";

private static ResourceBundle sResourceBundle =
  LayeredResourceBundle.getBundle(RESOURCE_NAME,
  atg.service.dynamo.LangLicense.getLicensedDefault());
```

29. Add the following code above the `getRetCodes` method:

```
private final int SUCCESS = 1;
protected int[] mRetCodes = { SUCCESS };
```

30. In the `getRetCodes` method, erase the `TODO` comment and replace `return null;` with `return mRetCodes;`.
31. In the `runProcess` method, change the method arguments from `arg0` and `arg1` to `pParam` and `pResult`.
32. In the body of the `runProcess` method, erase the `TODO` comment and `return 0;` and add the following code:

```
ValidatePaymentGroupPipelineArgs args =
 (ValidatePaymentGroupPipelineArgs)pParam;
```

```
Order order = args.getOrder();
PaymentGroup paymentGroup = args.getPaymentGroup();

if (paymentGroup == null)
  throw new atg.commerce.order.InvalidParameterException
  (ResourceUtils.getMsgResource(
  "InvalidPaymentGroupParameter", RESOURCE_NAME,
   sResourceBundle));

if (!(paymentGroup instanceof PointsPaymentGroup))
  throw new atg.commerce.order.InvalidParameterException
  (ResourceUtils.getMsgResource(
  "InvalidPaymentGroupParameter", RESOURCE_NAME,
  sResourceBundle));

if (isLoggingDebug())
  logDebug("Validating Points Payment Group for order "
  + order.getId());

//validate that this is a reward order
MyStoreOrder myOrder = (MyStoreOrder) order;

if (!myOrder.isReward()){
  pResult.addError(
  "pointsPaymentGroupNonRewardOrderError",
  "Points payment group in non-reward order");
  return SUCCESS;
}

//validate that points payment group has a points item
PointsPaymentGroup pointsPG =
 (PointsPaymentGroup) paymentGroup;

RepositoryItem pointsItem = pointsPG.getPointsItem();

if (pointsItem == null){
  pResult.addError(
  "pointsPaymentGroupNoPointsItemError",
  "Points payment group has no associated points item");
}
return SUCCESS;
```

33. Save the class and correct any errors.
34. Build project and switch to the ATG perspective and the ATG Component Browser. Create a new component under `/com/mystore/order/processor` named `ValidatePointsPaymentGroup`. The component should be based on the `com.mystore.order.processor.MyStoreProcValidatePointsPaymentGroup` class. Select the **Edit component after completion** box.
35. Set the new component's properties as follows and save the component:
    `loggingDebug` =`true`
    `loggingIdentifier` =`MyStoreProcValidatePointsPaymentGroup`

36. In a text editor or Eclipse, open `/atg/commerce/commercepipeline.xml` for editing. Add the following lines just above `</pipelinemanager>`:

```
<pipelinechain name="validatePaymentGroup">
  <pipelinelink name="dispatchOnPGType">
    <transition returnvalue="6000"
      link="validatePointsPG" />
  </pipelinelink>

  <pipelinelink name="validatePointsPG"
    transaction="TX_MANDATORY">
     <processor jndi="/com/mystore/order/processor/
       ValidatePointsPaymentGroup" />
```

(**Note:** The `jndi` value should be all on one line.)

```
 </pipelinelink>
</pipelinechain>

<pipelinechain
  name="validatePaymentGroupPreConfirmation">
  <pipelinelink name="dispatchOnPGTypePreConfirmation">
    <transition returnvalue="6000"
     link="validatePointsPGPreConfirmation" />
  </pipelinelink>

  <pipelinelink name="validatePointsPGPreConfirmation"
   transaction="TX_MANDATORY">
     <processor jndi="/com/mystore/order/processor/
       ValidatePointsPaymentGroup" />
```

(**Note:** The `jndi` value should be all on one line.)

```
  </pipelinelink>
</pipelinechain>
```

37. Open the `/atg/commerce/order/processor/ValidatePaymentGroupByType` component for editing. Right click the `returnValues` property and select edit. Select the last line and click **Add**. Enter `pointsPaymentGroup` as the key and `6000` as the value. Click OK
38. Click **OK** followed by **Save** to save the component configuration.
39. Rebuild project, assemble the `atg_production.ear` file, and deploy to WebLogic.
40. In the **Dynamo Admin UI** Component Browser, browse to `/atg/commerce/PipelineManager`. Click the link for its `definitionFile` property to view the combined version of `commercepipeline.xml`. Locate the `validatePaymentGroup` and `validatePaymentGroupPreConfirmation` pipelines and confirm that your changes have been included.
41. Navigate to the `/atg/commerce/order/processor/ValidatePaymentGroupByType` component and confirm that your changes have been made to the `returnValues` property.

Task 4: Adding Points Functionality to `BillingProcessHelper` and `BillingInfoFormHandler`

42. In Eclipse, switch to the Java EE perspective and create a new class named `MyStoreBillingProcessHelper` in the `com.mystore.order.purchase` package. This class should extend `atg.projects.store.order.purchase.StoreBillingProcessHelper`.
43. Add the following import statements:

```
import java.util.Iterator;
import java.util.List;
import atg.commerce.CommerceException;
import atg.commerce.order.*;
import atg.repository.RepositoryItem;
import com.mystore.order.*;
```

44. Create a method with the following signature:

```
public void initializePointsPaymentGroup (Order pOrder,
RepositoryItem pProfile) throws CommerceException {
  MyStoreOrder myOrder = (MyStoreOrder) pOrder;

   if (myOrder.isReward()) {
     PaymentGroupManager pgm = getPaymentGroupManager();
logDebug("profile id is " + pProfile.getRepositoryId());
     RepositoryItem pointsItem = (RepositoryItem)
pProfile.getPropertyValue("points");
     logDebug("points item id is " + pointsItem.getRepositoryId());

     if (pointsItem != null) {
       PointsPaymentGroup pointsPaymentGroup = null;

       //check for an existing points payment group

       List paymentGroups = pOrder.getPaymentGroups();

       Iterator pgIterator = paymentGroups.iterator();

       while (pgIterator.hasNext()) {
         PaymentGroup pg = (PaymentGroup) pgIterator.next();
         if (pg.getPaymentGroupClassType().equals
           ("pointsPaymentGroup"))
           pointsPaymentGroup = (PointsPaymentGroup) pg;
       } //end while
```

```
          //no existing group, so create new one
          if (pointsPaymentGroup == null){

            pointsPaymentGroup = (PointsPaymentGroup)
              pgm.createPaymentGroup("pointsPaymentGroup");

            pgm.addPaymentGroupToOrder(pOrder,
              pointsPaymentGroup, 0);

            pointsPaymentGroup.setPointsItem(pointsItem);
          } //end if pointsPaymentGroup is null

          double pointsNeeded = 0;

          Iterator itemIterator =
            myOrder.getCommerceItems().iterator();

          while (itemIterator.hasNext()) {
            CommerceItem item =
             (CommerceItem) itemIterator.next();

            pointsNeeded += item.getPriceInfo().getAmount();

          } //end while items
          Integer availableBalance = (Integer)
            pointsItem.getPropertyValue("availableBalance");

          if (pointsNeeded > availableBalance.doubleValue()) {
            throw new CommerceException
            ("Not enough points to make this purchase");
          } //end if not enough points

          String pgID = pointsPaymentGroup.getId();

   getPaymentGroupManager().removeAllRelationshipsFromPaymentGroup(
pOrder, pgID);
          getOrderManager().addOrderAmountToPaymentGroup(pOrder, pgID,
   pointsNeeded);

          } //end if pointsItem not null
      else {
```

```
      throw new CommerceException("User profile does not have
associated points item");
   } //else points item is null

} //end if order is reward
```

45. In Eclipse, switch to the Java EE perspective and create a new class named `MyStoreBillingInfoFormHandler` in the `com.mystore.order.purchase` package. This class should extend `atg.projects.store.order.purchase.BillingInfoFormHandler`.

46. Add the following import statements:

```
import java.io.IOException;
import javax.servlet.ServletException;
import com.mystore.order.MyStoreOrder;
import atg.commerce.CommerceException;
import atg.droplet.DropletFormException;
import atg.servlet.DynamoHttpServletRequest;
import atg.servlet.DynamoHttpServletResponse;
```

47. Add the following code:

```
public void preSetupStoreCreditPaymentGroupsForOrder
 (DynamoHttpServletRequest request,
  DynamoHttpServletResponse response)
 throws ServletException, IOException {

MyStoreBillingProcessHelper helper =
 (MyStoreBillingProcessHelper) getBillingHelper();
MyStoreOrder myOrder = (MyStoreOrder) getOrder();

try {
  if (myOrder.isReward())
   helper.initializePointsPaymentGroup
   (getOrder(),getProfile());
 } //end try

catch (CommerceException ce) {
  addFormException(new
    DropletFormException(ce.getMessage(),null));
}//end catch
} //end method
```

48. Use a text editor or Eclipse to create a file named `/atg/store/order/purchase/BillingProcessHelper.properties` under the **`MyStore.Custom`** `config` folder. Add the following line:

```
$class=\
com.mystore.order.purchase.MyStoreBillingProcessHelper
```

49. Use a text editor or Eclipse to create a file named `/atg/store/order/purchase/BillingFormHandler.properties` file under the **`MyStore.Custom`** `config` folder. Add the following line:

```
$class=\
com.mystore.order.purchase.MyStoreBillingInfoFormHandler
```

50. Rebuild project, assemble the `atg_production.ear` file, and deploy to WebLogic.
51. In the **ATG Store**, log out (you may need to click the **Logout** link twice) and log in as `eric@example.com` (password is `password`).
52. Switch to the **US Reward Site** and add a **Paris Clock** to your cart.
53. Click the **Checkout** link. On the cart page, click the **Checkout** button.
54. Eric's home address should be selected by default. Click the **Ship to this Address** button.
55. Select a shipping method and click **Continue**. The billing page will load. (In a real-world application, you would alter the display of the cart page to display the amount of the order covered by points, versus the shipping and taxes covered by the credit card.)
56. Switch to the **Dynamo Admin UI** Component Browser and use `GenericSessionManager` to view the current order in the context of the **US Reward Site**.
57. View the order's `paymentGroups` property. There should be two payment groups: a points group and a credit card group.
58. View the order's `paymentGroupRelationships` property. There should be two relationships. The first should allocate the cost of the clock to the `points` payment group. The second should be of type `ORDERAMOUNTREMAINING` and be associated with the credit card.

Task 5: Adding a Payment Group Payment Processor

59. In Eclipse, switch to the Java EE perspective and create a new class named `MyStoreProcProcessPointsPayment` in the `com.mystore.payment.processor` package. This class should extend `atg.commerce.payment.processor.ProcProcessPaymentGroup`.
60. Add the following import statements:

```
import atg.repository.*;
import atg.payment.PaymentStatusImpl;
```

61. Add a `MutableRepository` property called `profileRepository` with the appropriate get and set methods.
62. In the `authorizePaymentGroup` method, change the argument name from `arg0` to `pParams`. Erase the TODO comment and `return null;`. Do the same for the `creditPaymentGroup` and `debitPaymentGroup` methods.
63. Add the following code to the `authorizePaymentGroup` method:

```
double amount = pParams.getAmount();
```

```
String profileId = pParams.getOrder().getProfileId();

try{
 MutableRepository profileRep = getProfileRepository();
 RepositoryItem pointsItem = (RepositoryItem)
  profileRep.getItem(profileId,"user").getPropertyValue
  ("points");

 MutableRepositoryItem pointsItemMut =
  profileRep.getItemForUpdate
  (pointsItem.getRepositoryId(),"points");

 Integer reservedBalance = (Integer)
  pointsItem.getPropertyValue("reservedBalance");
 if (reservedBalance == null) reservedBalance = 0;

 Integer availableBalance = (Integer)
  pointsItem.getPropertyValue("availableBalance");
 if (availableBalance == null) availableBalance = 0;

 double zeroCheck =
  availableBalance.doubleValue() - amount;

 if (zeroCheck < 0.0) {
  return new PaymentStatusImpl(Long.toString
  (System.currentTimeMillis()), amount, false,
   "Points needed exceeds available balance",
   new java.util.Date());
 } //end if points exceeded balance

 Double newReservedBalance =
  new Double(reservedBalance.doubleValue() + amount);

 Double newAvailableBalance =
  new Double(availableBalance.doubleValue() - amount);

 pointsItemMut.setPropertyValue("reservedBalance",
 newReservedBalance .intValue());

 pointsItemMut.setPropertyValue("availableBalance",
 newAvailableBalance .intValue());

 profileRep.updateItem(pointsItemMut);
} //end try

catch (Exception exc) {
 String errorMsg = exc.getMessage() != null ?
  exc.getMessage() : exc.toString();

 return new PaymentStatusImpl(Long.toString
  (System.currentTimeMillis()), amount,false, errorMsg,
   new java.util.Date());
}//end catch
```

```
return new PaymentStatusImpl(Long.toString
 (System.currentTimeMillis()), amount, true, "",
 new java.util.Date());
```

64. Add the following code to the `creditPaymentGroup` method:

```
double amount = pParams.getAmount();
String profileId = pParams.getOrder().getProfileId();

try {
  MutableRepository profileRep = getProfileRepository();

  RepositoryItem pointsItem = (RepositoryItem)
   profileRep.getItem(profileId,"user").getPropertyValue
   ("points");

  MutableRepositoryItem pointsItemMut =
   profileRep.getItemForUpdate
   (pointsItem.getRepositoryId(),"points");

  Integer reservedBalance =
   (Integer)pointsItem.getPropertyValue
   ("reservedBalance");

  Integer availableBalance =
   (Integer)pointsItem.getPropertyValue
   ("availableBalance");

  Double newReservedBalance = new
   Double(reservedBalance.doubleValue() - amount);

  Double newAvailableBalance = new
   Double(availableBalance.doubleValue() + amount);

  pointsItemMut.setPropertyValue("reservedBalance",
   newReservedBalance.intValue());

  pointsItemMut.setPropertyValue("availableBalance",
   newAvailableBalance.intValue());

  profileRep.updateItem(pointsItemMut);
}//end try

catch (Exception exc) {
  String errorMsg = exc.getMessage() != null ?
   exc.getMessage() : exc.toString();

  return new PaymentStatusImpl(Long.toString
   (System.currentTimeMillis()), amount,false, errorMsg,
    new java.util.Date());
}//end catch
```

```
return new PaymentStatusImpl(Long.toString
 (System.currentTimeMillis()), amount, true, "", new
 java.util.Date());
```

65. Add the following code to the `debitPaymentGroup` method:

```
double amount = pParams.getAmount();
String profileId = pParams.getOrder().getProfileId();

try{
  MutableRepository profileRep = getProfileRepository();

  RepositoryItem pointsItem = (RepositoryItem)
   profileRep.getItem(profileId,"user")
   .getPropertyValue("points");

  MutableRepositoryItem pointsItemMut =
   profileRep.getItemForUpdate
   (pointsItem.getRepositoryId(),"points");

  Integer reservedBalance =
   (Integer)pointsItem.getPropertyValue
   ("reservedBalance");

  //decrease reservedBalance, not availableBalance,
  //because availableBalance was decreased in authorize

  Double newReservedBalance = new
   Double(reservedBalance.doubleValue() - amount);

  pointsItemMut.setPropertyValue("reservedBalance",
   newReservedBalance.intValue());

  profileRep.updateItem(pointsItemMut);
}//end try

catch (Exception exc) {
  String errorMsg = exc.getMessage() != null ?
   exc.getMessage() : exc.toString();

  return new PaymentStatusImpl(Long.toString
   (System.currentTimeMillis()), amount,false, errorMsg,
   new java.util.Date());
}//end catch
return new PaymentStatusImpl(Long.toString
  (System.currentTimeMillis()), amount, true, "", new
  java.util.Date());
```

66. Build project and switch to the ATG perspective and the ATG Component Browser. Create a new component under `/com/mystore/payment/processor` named `ProcessPointsPayment`. The component should be based on the

com.mystore.payment.processor.MyStoreProcProcessPointsPayment class. Select the **Edit component after completion** box.

67. Set the new component's profileRepository property to /atg/userprofiling/ProfileAdapterRepository and save the component.
68. Using a text editor or Eclipse, create a file named /atg/commerce/payment/paymentpipeline.xml under the **MyStore.Custom** config folder.
69. Add the following content to this file and save it:

```
<pipelinemanager>
 <pipelinechain name="pointsPaymentProcessorChain"
  transaction="TX_REQUIRED"
  headlink="processPointsPayment">

   <pipelinelink name="processPointsPayment"
    transaction="TX_MANDATORY">

   <processor jndi="/com/mystore/payment/processor/
    ProcessPointsPayment"/>
```

(**Note:** The jndi value should be all on one line.)

```
  </pipelinelink>
 </pipelinechain>
</pipelinemanager>
```

70. Use a text editor to create the /atg/commerce/payment/PaymentManager.properties file under the **MyStore.Custom** config folder. Add the following lines and save the file:

```
paymentGroupToChainNameMap+=\
com.mystore.order.PointsPaymentGroup=\
pointsPaymentProcessorChain
```

71. Rebuild project, assemble the atg_production.ear file, and deploy to WebLogic.
72. In the **Dynamo Admin UI** Component Browser, browse to /atg/commerce/payment/PaymentPipelineManager. Ensure that pointsPaymentProcessorChain is included in the list of pipelines.
73. Navigate to the /atg/commerce/payment/PaymentManager component and confirm that your changes have been made to the paymentGroupToChainNameMap property.
74. Open the home page of the **US Reward Site**. You should still be logged is as Eric, with a Paris Clock in your cart. Click the **Checkout** link.
75. On the cart page, click the **Checkout** button. You will be asked to log in again; the password is password.
76. On the shipping page, click the **Ship to the Address** button under the Home address. Select a shipping method and click **Continue**.
77. On the billing page, type 111 in the **CSV Code** field and click the **Continue** button.
78. On the confirmation page, click the **Place My Order** button.
79. If the order was placed successfully, note the order ID number given on the confirmation page. If not, check the Production server console for errors.

80. Start the **ACC** and navigate to **People & Organizations > Users**. Click the **List** button to view the users.
81. Find `eric@example.com`. Double-click his `points` property (where it says `UserProfiles:points:<id-number>`) to view his points object. The `availableBalance` should be reduced by the cost of the clock, which should now be reflected in the `reservedBalance`.
82. Navigate to **Purchases and Payments > Orders**. Click the List button to list the orders find the one with the ID number you noted earlier. Notice that the state of the order (depending on timing, it should be either `SUBMITTED` or `PROCESSING`). If it is `PROCESSING`, the order has been picked up by the Fulfillment module.
83. Go to the Dynamo Admin page and click the **Commerce Administration** link, then the **Fulfillment Administration** link.
84. Click the **Click here** link under the **List Shippable Groups** section. Your test order's shipping group should be listed there, because the Fulfillment system has now done its initial processing on the group (reserving inventory).
85. Pretend that you just shipped this order. Copy the order and shipping group IDs into the fields in the **Notify Fulfillment of shipping** section lower down on this page and click the **Ship…** button. This will trigger the Fulfillment system to debit the appropriate payment groups.
86. Open the **ACC** and navigate to **People & Organizations > Users**. Click the **List** button to view the users.
87. Find `eric@example.com`. Double-click his points property (where it says `UserProfiles:points:<id-number>`) to view his points object. The `reservedBalance` should now be `0`. Note that you may need to wait a minute for this update to take place.

Task 6: Extending Payment Processing to Award Points for Regular Orders

88. In Eclipse, switch to the Java EE perspective and create a new class named `MyStoreProcAwardPoints` in the `com.mystore.payment.processor` package. This class should extend `atg.nucleus.GenericService` and implement `atg.service.pipeline.PipelineProcessor`.
89. Add the following import statements:

```
import com.mystore.order.MyStoreOrder;
import com.mystore.pricing.MyStoreOrderPriceInfo;
import atg.commerce.payment.PaymentManagerAction;
import atg.commerce.payment.PaymentManagerPipelineArgs;
import atg.payment.PaymentStatus;
import atg.repository.MutableRepository;
import atg.repository.MutableRepositoryItem;
import atg.repository.RepositoryItem;
```

90. Add a `MutableRepository` property called `profileRepository` with the appropriate get and set methods.

91. Add the following code above the `getRetCodes` method:

```
private final int SUCCESS = 1;
protected int[] mRetCodes = { SUCCESS };
```

92. In the `getRetCodes` method, erase the TODO comment and replace `return null;` with `return mRetCodes;`.
93. In the `runProcess` method, change the method arguments from `arg0` and `arg1` to `pParam` and `pResult`.
94. In the body of the `runProcess` method, erase the TODO comment and `return 0;` and add the following code:

```
try {
 PaymentManagerPipelineArgs params =
 (PaymentManagerPipelineArgs)pParam;

 PaymentStatus paymentStatus =
 (PaymentStatus)params.getPaymentStatus();

 MyStoreOrder order = (MyStoreOrder)params.getOrder();

 PaymentManagerAction action = params.getAction();

 if (paymentStatus.getTransactionSuccess()
     && action == PaymentManagerAction.DEBIT
     && !order.isReward()) {

  MyStoreOrderPriceInfo priceInfo =
  (MyStoreOrderPriceInfo) order.getPriceInfo();

  int points = priceInfo.getPoints();

  String profileId = order.getProfileId();

  MutableRepository profileRep = getProfileRepository();

  RepositoryItem pointsItem = (RepositoryItem)
   profileRep.getItem(profileId,"user").getPropertyValue
   ("points");

  MutableRepositoryItem pointsItemMut = null;

  if (pointsItem == null)
   pointsItemMut = profileRep.createItem("points");

  else
   pointsItemMut = profileRep.getItemForUpdate
   (pointsItem.getRepositoryId(),"points");

  Integer availableBalance = (Integer)
   pointsItem.getPropertyValue("availableBalance");
```

```
    if (availableBalance == null) availableBalance = 0;

    int newAvailableBalance = availableBalance + points;

    pointsItemMut.setPropertyValue("availableBalance",
     newAvailableBalance);

    profileRep.updateItem(pointsItemMut);
   } //end if qualifying transaction
  } //end try

  catch(Exception e) {
   logError(e);
   pResult.addError("ProcProcessPaymentGroupFailed", e);
   return STOP_CHAIN_EXECUTION;
  } //end catch

  return SUCCESS;
```

95. Build project and switch to the ATG perspective and the ATG Component Browser. Create a new component under `/com/mystore/payment/processor` named `AwardPoints`. The component should be based on the `com.mystore.payment.processor.MyStoreProcAwardPoints` class. Select the **Edit component after completion** box.
96. Set the new component's properties as follows and save the component: `profileRepository` =`/atg/userprofiling/ProfileAdapterRepository`
97. In a text editor or Eclipse, edit the `/atg/commerce/payment/paymentpipeline.xml` file that you created earlier. Add the following lines just above the `</pipelinemanager>` tag:

```
<pipelinechain name="creditCardProcessorChain">
 <pipelinelink name="processCreditCard">
  <transition link="awardPoints" returnvalue="1"/>
 </pipelinelink>

 <pipelinelink name="awardPoints"
   transaction="TX_MANDATORY">
  <processor
   jndi="/com/mystore/payment/processor/AwardPoints"/>
  </pipelinelink>
</pipelinechain>
```

98. Rebuild the project, assemble the `atg_production.ear` file, and deploy to WebLogic.
99. In the **Dynamo Admin UI** Component Browser, browse to `/atg/commerce/payment/PaymentPipelineManager`. Click the link for its `definitionFile` property to view the combined version of `paymentpipeline.xml`. Locate the `creditCardProcessorChain` pipeline and confirm that your changes have been included.
100. Open the **ATG Store**. You should still be logged in as Eric.

101. Add any regular item to your cart and check your order out as you did with the reward order. After you have submitted the order, note the order ID.
102. Open the **ACC** and navigate to **Purchases and Payments > Orders**. Click the List button to list the orders, find the one with the ID you noted earlier. Double-click its Price Info value to view its price info object. Note the number of points earned for this order.
103. Go to the home page of the browser-based admin interface and click the **Commerce Administration** link, then the **Fulfillment Administration** link.
104. Click the link under the **List Shippable Groups** section. If your order is not listed, wait a minute and refresh this page. Copy the order and shipping group IDs of your order into the fields in the **Notify Fulfillment of shipping** section lower down on this page and click the **Ship…** button.
105. In the **ACC**, navigate to **People & Organizations > Users**. Click the **List** button to view the users.
106. Find `eric@example.com`. Double-click his points property to view his points object.

The `availableBalance` should include the amount earned for the test order.

Practices for Lesson 18: Other Purchase Process Form Handlers

Chapter 18

Practices for Lesson 18

Practices Overview

There are no practices for this lesson.

Practices for Lesson 19: Inventory and Fulfillment

Chapter 19

Practices for Lesson 19

Practices Overview

There are no practices for this lesson.