Introduction to Purchase Flow

Presenter's Name
Presenter's Title

Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions.
The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Oracle Training Materials – Usage Agreement

Use of this Site ("Site") or Materials constitutes agreement with the following terms and conditions:

1. Oracle Corporation ("Oracle") is pleased to allow its business partner ("Partner") to download and copy the information, documents, and the online training courses (collectively, "Materials") found on this Site. The use of the Materials is restricted to the non-commercial, internal training of the Partner's employees only. The Materials may not be used for training, promotion, or sales to customers or other partners or third parties.

2. All the Materials are trademarks of Oracle and are proprietary information of Oracle. Partner or other third party at no time has any right to resell, redistribute or create derivative works from the Materials.

3. Oracle disclaims any warranties or representations as to the accuracy or completeness of any Materials. Materials are provided "as is" without warranty of any kind, either express or implied, including without limitation warranties of merchantability, fitness for a particular purpose, and non-infringement.

4. Under no circumstances shall Oracle or the Oracle Authorized Delivery Partner be liable for any loss, damage, liability or expense incurred or suffered which is claimed to have resulted from use of this Site of Materials. As a condition of use of the Materials, Partner agrees to indemnify Oracle from and against any and all actions, claims, losses, damages, liabilities and expenses (including reasonable attorneys' fees) arising out of Partner's use of the Materials.

5. Reference materials including but not limited to those identified in the Boot Camp manifest can not be redistributed in any format without Oracle written consent.

Agenda

- Form Handlers - Modifying Cart
- Form Handlers - Checking Out

Learning Objectives

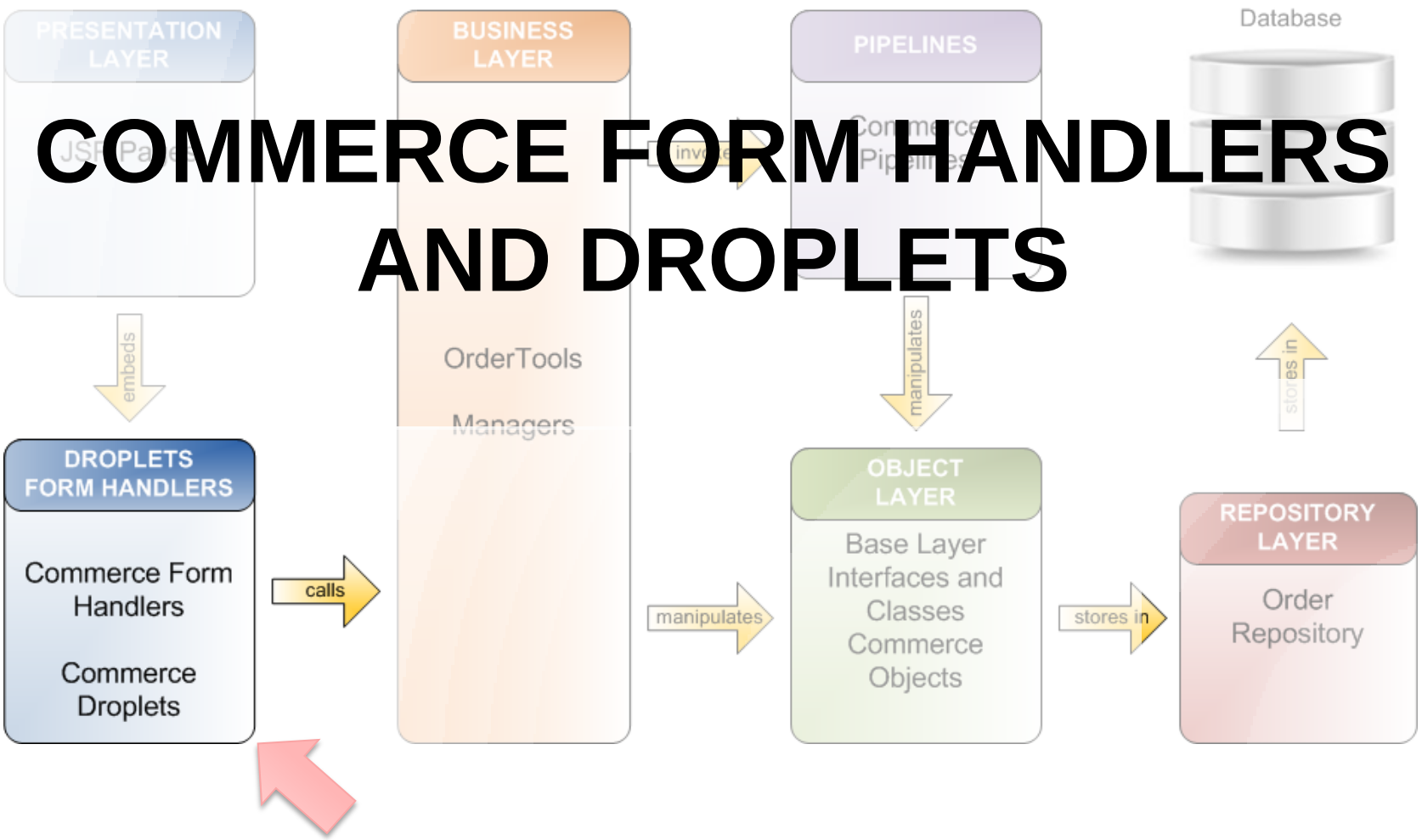
At the end of this lesson you should be able to:

- Learn about the various commerce form handlers and droplets
- Use the cart modifier form handler to add items to cart and modify current order
- Use checkout form handler and express checkout form handler to place an order
- Create shipping groups with the Shipping group form handler
- Create payment groups with the Payment group form handler
- Submit, save and cancel and order
- Learn about the Full Shopping Cart form handler to place an order

Section 1:
Commerce Form Handlers and Droplets

Modifying Cart

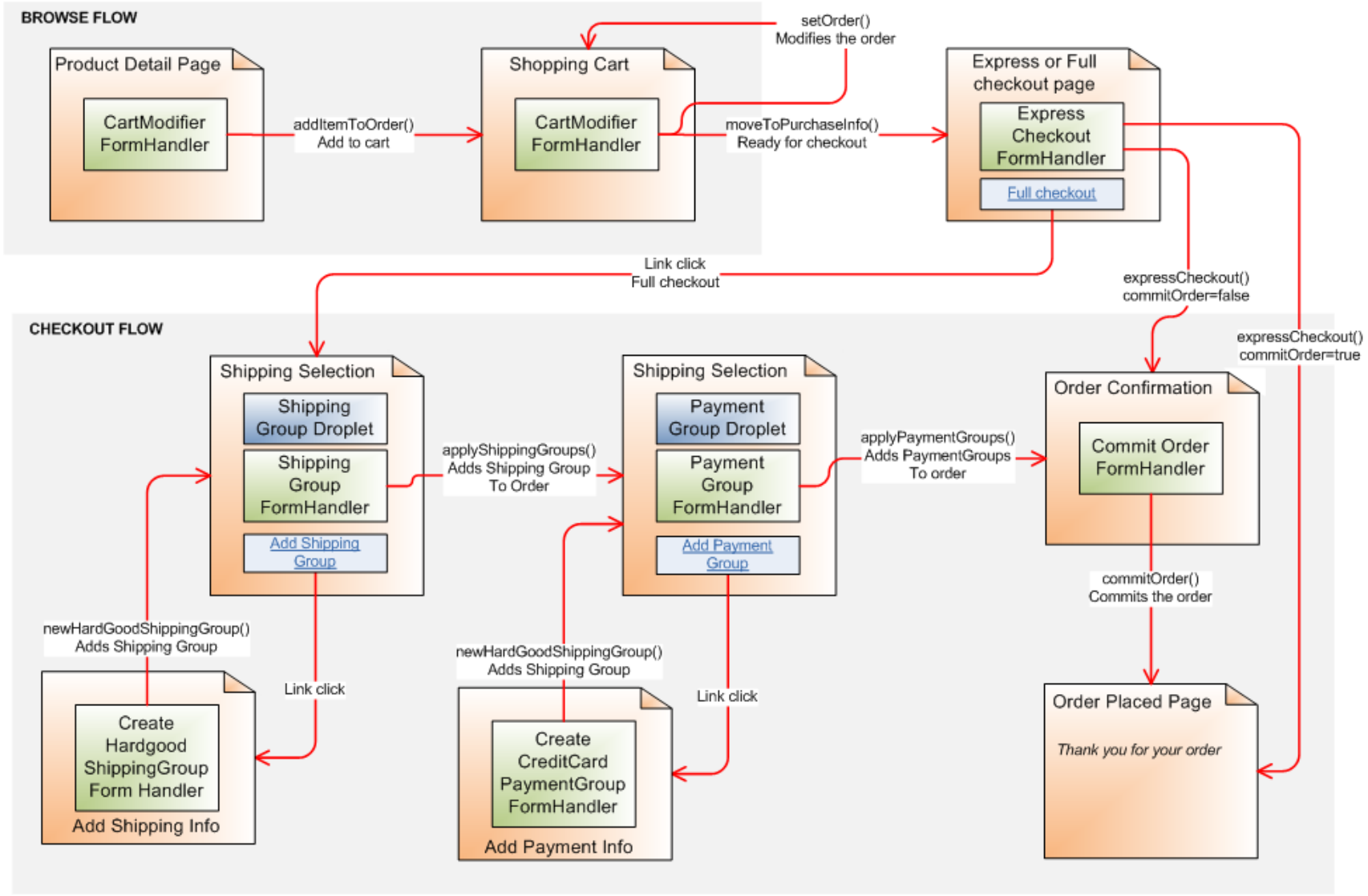
COMMERCE FORM HANDLERS AND DROPLETS

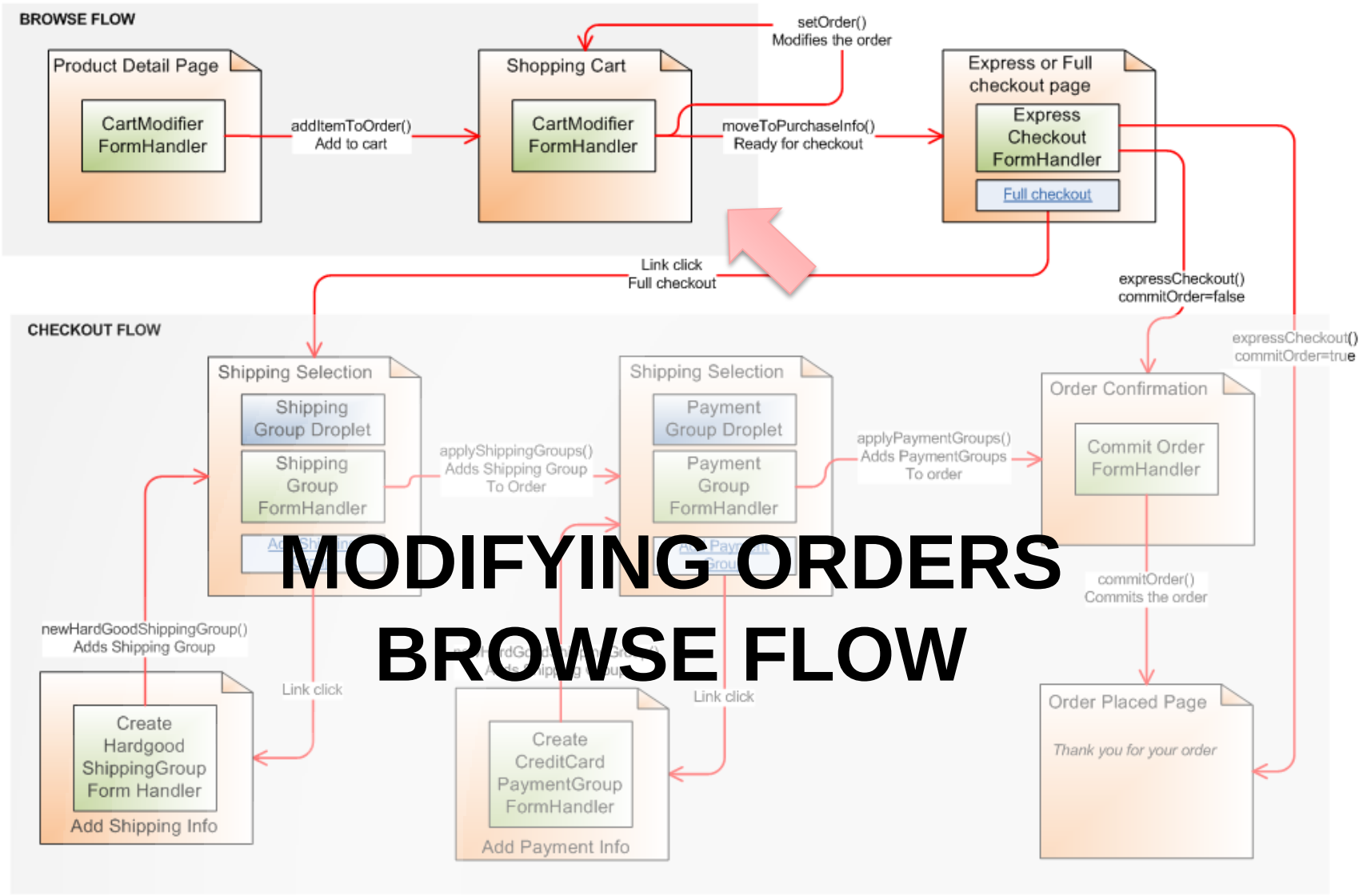


Commerce Form Handlers

- The commerce form handlers simplify the work of managing commerce objects by exposing management functionality. They call the manager classes internally.
- The following are commerce form handlers:
  - **CartModifierFormHandler** is used to add to and modify the cart.
  - **ExpressCheckoutFormHandler** is used to express checkout.
  - **ShippingGroupFormHandler** associates shipping groups with order.
  - **PaymentGroupFormHandler** associates payment groups with order.
  - **CommitOrderFormHandler** commits the order and submits it.
  - **SaveOrderFormHandler** is used to save an order and create a new blank order.
  - **CancelOrderFormHandler** cancels the order and deletes it.
  - There are other form handlers for creating shipping and payment groups.

Commerce Form Handler Actions





MODIFYING ORDERS BROWSE FLOW

Understanding the CartModifierFormHandler

- The CartModifierFormHandler provides:
  - Add items to an order,
  - Remove items from an order,
  - Modify the quantity of items in the order,
  - Prepare the order for the checkout process.
- CartModifierFormHandler is an instance of class atg.commerce.order.purchase.CartModifierFormHandler; it is located in nucleus at /atg/commerce/order/purchase/CartModifierFormHandler.
- It provides many methods to change the order.
- It calls OrderManager.updateOrder() to save the order.

Add Items to Cart with the CartModifier FormHandler

- Use the **`handleAddItemToOrder`** method to add items to the cart.
- This method calls OrderManager.updateOrder() which calls the updateOrder pipeline chain.
- Developers need to pass:
  - addItemToOrderSuccessURL,
  - productId,
  - catalogRefIds,
  - Quantity,
  - **addItemToOrder** in the submit input tag.

Example Add Items to Cart

```
<dsp:form action="display_product.jsp" method="post">
  <input name="id" type="hidden"
       value='<dsp:valueof param="product.repositoryId"/>'>
  <dsp:input bean="CartModifierFormHandler.addItemToOrderSuccessURL"
                    type="hidden" value="shoppingcart.jsp"  />
  <dsp:input bean="CartModifierFormHandler.productId"
         paramvalue="product.repositoryId" type="hidden"/>
  <dsp:select bean="CartModifierFormHandler.catalogRefIds">
    <dsp:droplet name="/atg/dynamo/droplet/ForEach">
      <dsp:param name="array" param="product.childSKUs"/>
      <dsp:param name="elementName" value="sku"/>
      <dsp:param name="indexName" value="skuIndex"/>
      <dsp:oparam name="output">
         <dsp:option paramvalue="sku.repositoryId"/>
         <dsp:valueof param="sku.displayName"/>
      </dsp:oparam>
    </dsp:droplet>
  </dsp:select>
  Quantity: <dsp:input bean="CartModifierFormHandler.quantity"
                   size="4" type="text" value="1"/><BR>
  <dsp:input bean="CartModifierFormHandler.addItemToOrder"
                    type="submit" value="Add To Cart"/>
</dsp:form>
```

Modifying the Current Order

- To modify an order, you must supply either a CatalogRefId of a CommerceItem or a ShippingGroupCommerceItemRelationship ID.
- Modify order by CatalogRefId (SKU ID):
  - When you have a simple cart page.
  - When you don’t have multiple shipping groups like to support split shipment.
- Modify order by ShippingGroupCommerceItem relationship ID:
  - If you intend to support complex product-SKU relationships.
  - If you need the granularity to delete just a part of a CommerceItem.
  - If you intend to support multiple commerce items with the same catalog Ref ID.

Functions Modifying the Current Order

- To modify order by CatalogRefId, use:
  - **`handleSetOrder`**() Performs the actual work necessary to save an Order. This is for Updating orders on the cart page.
  - **`handleRemoveItemFromOrder`**() Removes items from the Order by CommerceItem ID.
  - **`handleMoveToPurchaseInfo`**() Performs the actual work necessary to save an Order and verifies that the order is ready for checkout.
- To modify order by shipping relationship, use:
  - handleSetOrderByRelationshipId().
  - handleRemoveItemFromOrderByRelationshipId().
  - handleMoveToPurchaseInfoByRelId().
- The ByRelationshipId methods do the same as the regular methods but accept relationship ID instead of catalog Ref ID.

Example of Modifying Order (1)

```
<dsp:form action="shoppingcart.jsp" method="post">
  <dsp:input type="hidden" value="purchase_info.jsp"
    bean="CartModifierFormHandler.moveToPurchaseInfoByRelIdSuccessURL" />
  <dsp:droplet name="ForEach">
    <dsp:param name="array"
                bean="CartModifierFormHandler.Order.ShippingGroups" />
    <dsp:param name="elementName" value="ShippingGroup"/>
    <dsp:oparam name="output">
      Shipping Group: <dsp:valueof param="ShippingGroup.Id" />
      <dsp:droplet name="ForEach">
        <dsp:param name="array"
                    param="ShippingGroup.CommerceItemRelationships"/>
      <dsp:param name="elementName" value="CiRelationship"/>
      <dsp:oparam name="output">
          SKU: <dsp:valueof
    param="CiRelationship.commerceItem.auxiliaryData.catalogRef.displayName"/>
          Delete: <dsp:input paramvalue="CiRelationship.Id" type="checkbox"
  bean="CartModifierFormHandler.removalRelationshipIds" checked="<%=false%>"/>
          Quantity: <input name='<dsp:valueof param="CiRelationship.Id"/>'
             size="4" value='<dsp:valueof param="CiRelationship.quantity"/>'>
          <hr>
  </dsp:oparam></dsp:droplet><hr><br/></dsp:droplet>
  <dsp:input bean="CartModifierFormHandler.setOrderByRelationshipId"
              type="submit" value="Recalculate"/>    
  <dsp:input bean="CartModifierFormHandler.moveToPurchaseInfoByRelId"
              type="submit" value="Checkout"/>
</dsp:form>
```

Example of Modifying Order (2)

- The procedure shown in the last example is:
  - Iterated over shipping groups.
    - For each shipping group, iterated over relationships.
      - For each relationship, you can set:
        - **`removalRelationshipIds`** check box for removing, and
        - Input with name as relationship ID and value as quantity.
  - Call **`setOrderByRelationshipId`** to recalculate the order, or
  - Call **`moveToPurchaseInfoByRelId`** to go to the checkout process.
- **`moveToPurchaseInfoByRelId`** iterates over all the shipping group relationships and verifies that they are ready for checkout.
- It then calls OrderManager.updateOrder() which calls the updateOrder pipeline chain.

Section 1
Check Your Understanding

Name some methods of CartModifierFormHandler.

Answer:

addItemsToOrder(), setOrder(), removeItemFromOrder(), and moveToPurchaseInfo(), etc.

Section 1 ✔
Check Your Understanding

Why would a developer need to modify by relationship ID instead of by catalogRefID (SKU ID)?

Answer:

If they support split shipping or complex product-SKU relationships.

Section 1

Check Your Understanding

When you are updating a user's cart based on modification to quantity, which method in the CartModifierFormHander should you call?

Answer:

handleSetOrder() method.

Section 1 ✔
Check Your Understanding

Name a few commerce form handlers.

Answer:

CartModifierFormHandler, ExpressCheckoutFormHandler, ShippingGroupFormHandler, etc.

Section 1 
Check Your Understanding

What are the four functions of the CartModifierFormHandler?

Answer:

Add, remove, modify items in order, and prepare for checkout process.

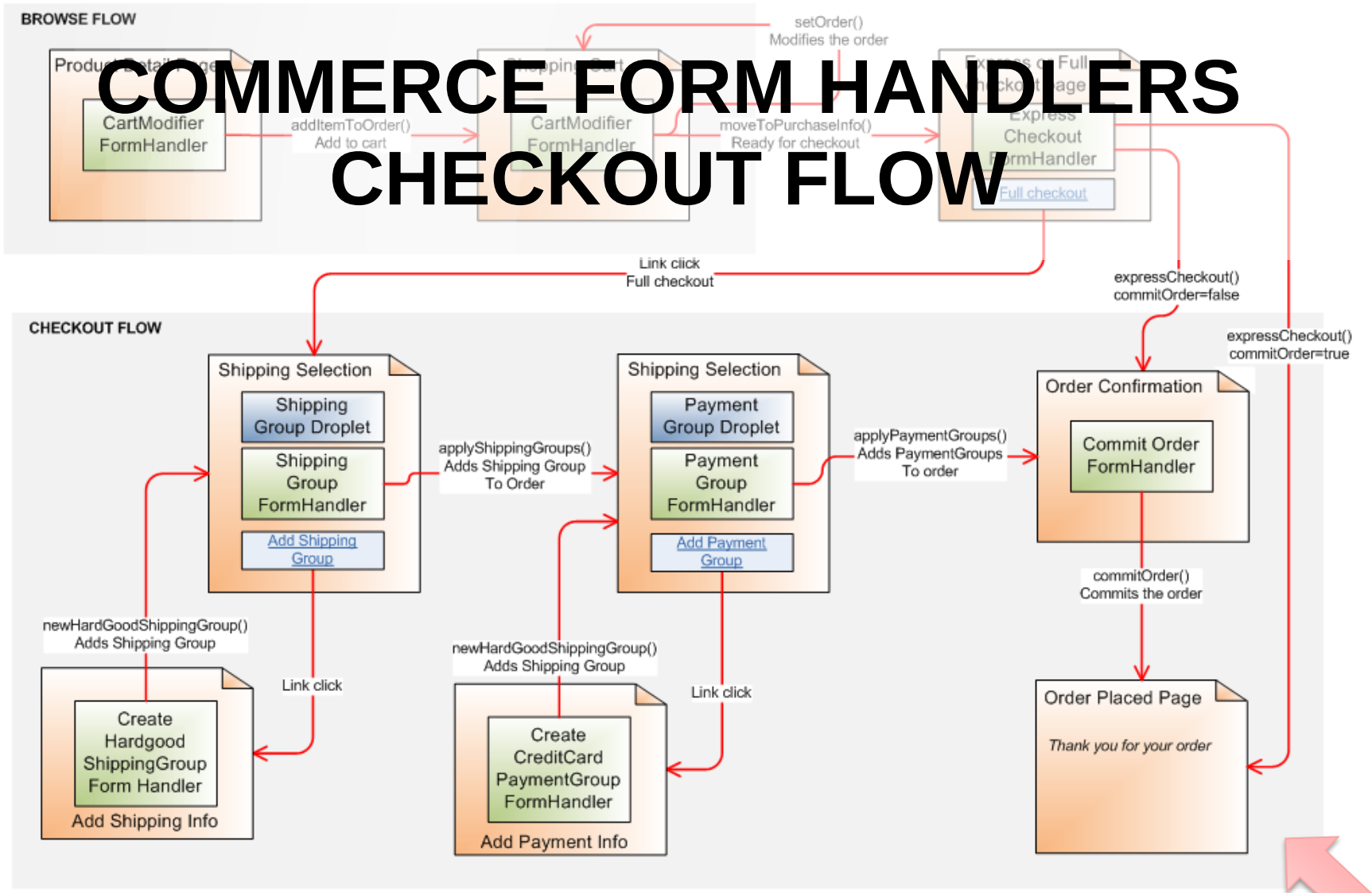
Summary

- The commerce form handlers simplify the work of managing commerce objects by exposing management functionality. They call the manager classes internally.
- The CartModifierFormHandler provides capability to add, remove, modify items in cart, and can prepare the order for checkout.
- You can manipulate items by CatalogRefID(SKU ID) or by relationship ID.
- Developers can use addItemToOrder to add to the cart.
- setOrder(), removeItemFromOrder(), and moveToPurchaseInfo() are other useful methods.
- There are equivalent methods to execute by RelationshipID.

Section 2:
Commerce Form Handlers and Droplets

Checking Out Order

COMMERCE FORM HANDLERS CHECKOUT FLOW



Checking Out Orders

- The order checkout process can vary depending on the requirements and complexities involved.
- ATG supports two types of checkout:
  - **Express checkout** only supports single shipping group and single payment group.
  - **Complex checkout** supports any number or type of shipping groups and/or payment groups.
- A site might implement both checkouts giving users the choice at the time of check out.
- Express checkout is sometimes referred to as '**One Click checkout.**'
- Typically, an express checkout is **only** available to registered users with shipping and payment method defaults already set in their profile.

Preparing a Simple Order for Checkout

- ExpressCheckoutFormHandler supports the use of only a single HardgoodShippingGroup and a single CreditCard for a given order.
- It can be used to manage and expedite the pre-checkout process for orders.
- The ShippingGroup and PaymentMethod should come from the user profile.
- The important methods and properties:
  - **`handleExpressCheckout`** is the main method.
  - **`paymentGroupNeeded`** if false will allow the user to supply a payment method from the profile.
  - **`shippingGroupNeeded`** if false will allow the user to supply shipping group information.
  - **`commitOrder`** if false can be used to not commit the order and display a order confirmation page.

Usage of Express Checkout FormHandler

- Typical sites will set **`commitOrder`** property to false.
- This allows for an order confirmation page to be displayed.
- The user credit card's CCV2 number, not typically stored, can be collected at this stage.
- Use the commitOrderFormHandler or the ExpressCheckoutFormHandler to commit the order.

```
<dsp:form action="exp_checkout.jsp" method="POST">
  <dsp:input type="hidden" value="orderConfirm.jsp"
       bean="ExpressCheckoutFormHandler.expressCheckoutSuccessURL" />
  <dsp:input type=hidden" value="false"
       bean="ExpressCheckoutFormHandler.commitOrder" />
  <dsp:input type="submit" value="Checkout"
       bean="ExpressCheckoutFormHandler.expressCheckout" />
</dsp:form>
```

Why Use Complex Order Checkout

- Complex checkout process supports the use of any number of type of shipping groups and payment groups.
- You would use complex order checkout:
  - If your customer does not have a profile (not a registered user).
  - If your customer does not want to use shipping and payment options already saved in the profile.
  - If you want to support split shipping (shipping to multiple addresses).
  - If you want to support split payment (paying with multiple payment methods).
  - If your payment method or shipping method requires front end integration (Paypal, Google Checkout).
- Most typical sites will implement complex checkout. It is also common to implement express checkout in parallel.

Preparing a Complex Order for Checkout

- ATG Commerce provides several form handlers to support a checkout process that uses any number or type of shipping groups and payment groups.
- The following are the difference sub processes in the preparation of an order for checkout:
  - Creating shipping groups,
  - Associating shipping groups with an order and its items,
  - Creating payment groups,
  - Associating payment groups with an order and its items.
- There are form handlers each of the above tasks.
- The order will then be ready to commit.

Creating Shipping Groups

- CreateShippingGroupFormHandler is an interface to support form driven creation of hardgood and electronic groups.
- The two default implementations of this interface are:
  - CreateHardgoodShippingGroupFormHandler,
  - CreateElectronicShippingGroupFormHandler.
- Once created, you can use the following to handle shipping address changes:
  - UpdateHardgoodShippingGroupFormHandler,
  - UpdateElectronicShippingGroupFormHandler.

Example of Create Hardgood ShippingGroup FormHandler

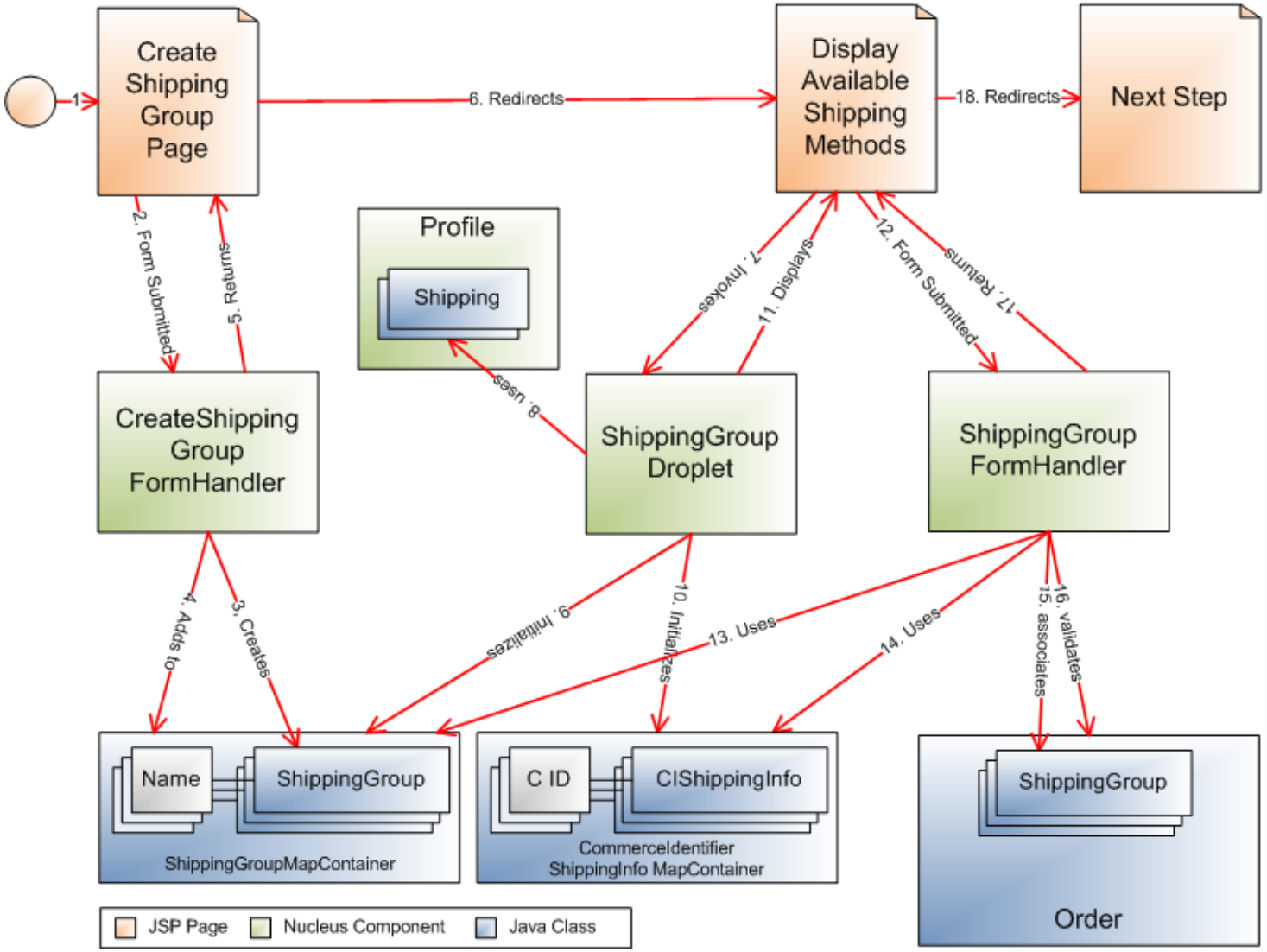
- The **`handleNewHardgoodShippingGroup`** method creates the shipping groups.
- In addition to the shown properties, you must also provide address2, city, state, postalCode and country.

```
<dsp:form action="hardgood_sg.jsp" method="post">
 ShippingGroup NickName: <dsp:input size="30" type="text" value="“
  bean="CreateHardgoodShippingGroupFormHandler.hardgoodShippingGroupName”/>
 <br/>
 First: <dsp:input beanvalue="Profile.firstName" type="text“
  bean="CreateHardgoodShippingGroupFormHandler.HardgoodShippingGroup
     .ShippingAddress.firstName" /> <br/>
  . . . <!-- address2, city, state, postalCode and country -->
 <br>
 <dsp:input bean="CreateHardgoodShippingGroupFormHandler
  .newHardgoodShippingGroupSuccessURL" type="hidden" value="shipping.jsp"/>
 <dsp:input type="submit" value="Create HardgoodShippingGroup“
  bean="CreateHardgoodShippingGroupFormHandler.newHardgoodShippingGroup" />
</dsp:form>
```

Associating Shipping Groups

- **ShippingGroupFormHandler** can be used to create and manage the associations between the ShippingGroups and the items in the order.
- This form handler works in conjunction with **ShippingGroupDroplet** to manipulate the relationships between CommerceItems and ShippingGroups in the order.
- The form handler uses the following containers:
  - **ShippingGroupMapContainer** defines a map of user-assigned ShippingGroup names to ShippingGroups.
  - **CommerceItemShippingInfoContainer** defines a map of CommerceItems to CommerceItemShippingInfo Lists.

Associating Shipping Groups Flow



Associating Shipping Groups Steps (1)

1. User visits the page to create shipping info (address and method).
2. User enters information to create shipping group such as address and shipping method and submits the form to the CreateShippingGroupFormHandler implementation (such as CreateHardgoodShippingGroupFormHandler).
3. 4., and 5. It creates the ShippingGroup and optionally associates with ShippingGroupMapContainer and returns.
6. The user is redirected to a page where the user can pick the shipping group. Rendering starts.
7. During rendering, the ShippingGroupDroplet is invoked.
8. The ShippingGroupDroplet accesses the payment methods stored on the profile.

Associating Shipping Groups Steps (2)

9. Intializers initialize the shipment groups and add them to ShipmentGroupMapContainer of the order.
10. It initializes the CommerIdentifier ShippingInfoMapContainer.
11. It finally displays the available shipping groups as output params and options to the user.
12. User selects the shippingGroup and submits the form.
13. and 14. ShippingGroupFormHandler is invoked. It uses the input and the two maps to identify the shippinggroup.
15. and 16. It then associates the shipping group to the order and its components creating the relationships as needed, and validates the order.
17. and 18. The form handler returns and request is forwarded to the next step.

ShippingGroupFormHandler

- With the help of the containers and helper classes, ShippingGroupFormHandler:
  - Establishes the relationships to the CommerceItems,
  - Performs validation and updates the order.
- The following methods are available:
  - **`handleApplyShippingGroups`** adds the ShippingGroups to the order. It is used to proceed to the next checkout phase.
  - **`handleSpecifyDefaultShippingGroup`** method is used to let the user specify a default ShippingGroup to use for shipping.
  - **`handleSplitShippingInfos`** method splits the quantities of CommerceItems across several CommerItemShippingInfos.

ShippingGroupDroplet

- ShippingGroupDroplet is used to display ShippingGroups to the user and allows the user to select them.
- ShippingGroupDroplet is used to initialize CommerceItemShippingInfo objects and add them to the CommerItemShippingInfoContainer.

```
<dsp:droplet name="ShippingGroupDroplet">
   <dsp:param name="clear" value="true" />
   <dsp:param name="shippingGroupTypes" value="hardgoodShippingGroup" />
   <dsp:param name="initShippingGroups" value="true" />
   <dsp:param name="initShippingInfos" value="true" />
   <dsp:oparam name="output">
      . . . Manipulation of objects here . . .
   </dsp:output>
</dsp:droplet>
```

Example of Associating Shipping Groups

```
<dsp:droplet name="ShippingGroupDroplet">
  <dsp:param name="clear" param="init"/>
  <dsp:param name="shippingGroupTypes" value="hardgoodShippingGroup"/>
  <dsp:param name="initShippingGroups" param="init"/>
  <dsp:param name="initShippingInfos"  param="init"/>
  <dsp:oparam name="output">
  <dsp:form action="shipping.jsp" method="post">
     <dsp:input type="hidden" value="billing.jsp?init=true"
       bean="ShippingGroupFormHandler.applyShippingGroupsSuccessURL" />
     <dsp:input type="hidden" value="complex_shipping.jsp?init=true"
bean="ShippingGroupFormHandler.specifyDefaultShippingGroupSuccessURL" />
     <dsp:input type="hidden" value="true"
       bean="ShippingGroupFormHandler.applyDefaultShippingGroup" />
       <b>Associate ShippingGroup</b><BR>
       <dsp:droplet name="ForEach">
         <dsp:param name="array" param="shippingGroups"/>
         <dsp:oparam name="output">
            <br><dsp:input paramvalue="key" type="radio"
 bean="ShippingGroupDroplet.ShippingGroupMapContainer.
defaultShippingGroupName" />
             <dsp:valueof param="key"/>
          </dsp:oparam>
        </dsp:droplet>
      <dsp:input bean="ShippingGroupFormHandler.applyShippingGroups"
          type="submit" value="Ship Entire Order to this Address"/>
  </dsp:form></dsp:oparam>
</dsp:droplet>
```

Creating Payment Groups

- CreatePaymentGroupFormHandler interface supports form driven creation of credit card and invoice payment groups.
- There are two implementations:
  - CreateCreditCardFormHandler is a form handler used to create CreditCard payment group.
  - CreateInvoiceRequestFormHandler is used for B2B to create an InvoiceRequest payment group.
- Once created, the payment group can be updated using:
  - UpdateCreditCardFormHandler.
- Once created, the payment groups can be added to PaymentGroupMapContainer.
- The user can then select from among the PaymentGroups in it.

Example of CreateCreditFormHandler

- The handleNewCreditCard method of the form handler accepts user's input and creates the CreditCardPaymentGroup.
- User has to provide:
  - Credit Card information,
  - Billing Address.

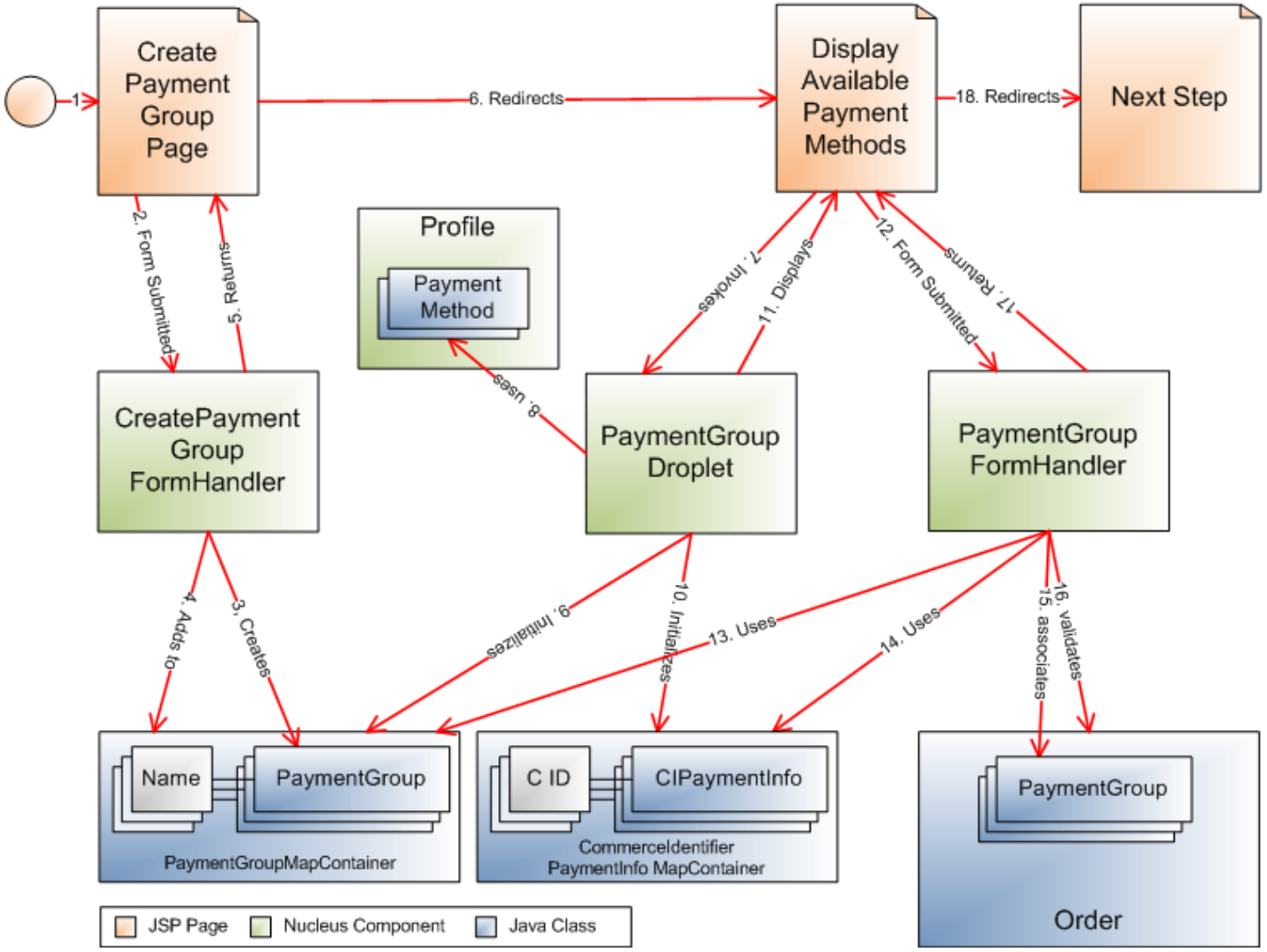
```
<dsp:form method='POST' action='paymentGroup.jsp'
                              id='createNewPGbyPMForm' >
  <jsp:include page="includes/payments/newCreditCard.jsp"/>
  <jsp:include page="includes/payments/billingAddress.jsp"/>

  <input type='hidden' name='pmId' id="pmId"/>
  <input type='hidden' name='pmMethod' id="pmMethod"/>
  <dsp:input bean="CreateCreditCardFormHandler.newCreditCard"
        priority="-10" type="hidden" value="submit" />
</dsp:form>
```

Associating Payment Groups

- PaymentGroupFormHandlers create and manage the associations between user supplied paymentGroups and the various parts of the order.
- PaymentGroupFormHandler:
  - Adds the PaymentGroups to the order,
  - Adds the CommerceItems, ShippingGroups, tax, cost amount, and cost remaining to the PaymentGroups,
  - Validates the PaymentGroup information,
  - Saves the order to the order repository.
- The PaymentGroupMapContainer contains a map of user assigned PaymentGroup names to the PaymentGroups.
- CommerceIdentifierPaymentInfoContainer defines a map of CommerceIdentifiers to CommerceIdentifierPaymentInfo Lists.

Associating Payment Groups Flow



Submitting an Order for Checkout

- The CommitOrderFormHandler submits the user's current order for checkout.
- The **`handleCommitOrder`**() commits the order.
  - Method checks the ID of the order to ensure that the user is not double submitting the order.
  - It calls **`OrderManager.processOrder`**() which executes the processOrder pipeline.
- If successful, it sets the submitted order as the user's last order and constructs a new, empty order which is set as the users current order.

```
<dsp:form action="order_confirmation.jsp" method="post">
  <dsp:input bean="CommitOrderFormHandler.commitOrderSuccessURL"
        type="hidden" value="orderPlaced.jsp"/>
  <dsp:input bean="CommitOrderFormHandler.commitOrder"
        type="submit" value="Purchase"/>
</dsp:form>
```

Re-pricing an Order

- The CartModifierFormHandler and ExpressCheckoutFormHandler automatically re-price an order when submitted.
- The ShippingFormHandler and PaymentFormHandler do not re-price an Order.
- You can use the RepriceOrderDroplet servlet bean to re-price an order.
  - When a custom promotion depends on the chosen shipping or payment method, you need to re-price the order.
- The RepriceOrderDroplet servlet bean will execute repriceAndUpdateOrder pipeline to re-price the order:

```
<dsp:droplet name="RepriceOrderDroplet">
  <dsp:param name="pricingOp" value="ORDER_SUBTOTAL" />
</dsp:droplet>
```

Saving Orders

- The SaveOrderFormHandler saves the user's current order and adds the order to the ShoppingCart's list of saved orders.
- Additionally, it constructs a new, empty order and sets it as the user's current order.
- ATG ships with a nucleus component located at /atg/commerce/order/purchase/SaveOrderFormHandler.
- The following are the important methods:
  - **`handleSaveOrder`** saves the order and creates a new empty order.
  - The order is saved based on description or date and time, if no description is provided.

Canceling Orders

- The CancelOrderFormHandler cancels the user's current order.
- If the order is in a state that can be deleted as configured in the deleteStates property, it is deleted. Otherwise, it is preserved.
- Component location is /atg/commerce/order/purchase/CancelOrderFormHandler.
- The important method in CancelOrderFormHandler is:
  - **`handleCancelOrder`** either deletes or preserves the Order based on its current state. The user is provided with a new, empty order.

ShoppingCartFormHandler

- The ShoppingCartFormHandler is used to control many aspects of the user purchase process. It provides:
  - Cart Management (adding, removing, adjusting quantity).
  - Checkout Process (shipping, billing, committing).
- This form handler provides an easier interface for order editing and checkout process than the form handlers already discussed.
- Functionality is exposed via various handleXXX methods.
- To extend the functionality, each handleXXX has a preXXX and postXXX.

FullShoppingCartFormHandler

- The FullShoppingCartFormHandler extends the functionality of ShoppingCartFormHandler and adds:
  - Handling multiple payment methods (gift certs, credit cards).
  - Split shipping.
  - Express checkout.
  - Adding an item to a person's gift list.
- It follows the same handleXXX, preXXX, and postXXX format of the ShoppingCartFormHandler.
- FullShoppingCartFormHandler can be an alternative to a simpler checkout implementation for some sites requiring more complex features.

Next Steps in the Order

- Order processing occurs when a customer has supplied all the necessary information for the order and has submitted it for checkout.
- The OrderManager calls the processOrder pipeline.
- The processOrder first validates the order and then processes it.
- It runs a set of pipeline processors.

Section 2 
Check Your Understanding

What are the two types of checkouts ATG supports?

Answer:

ATG supports: express checkout and complex checkout.

Section 2

Check Your Understanding

What parts of the order checkout process can ShoppingCartFormHandler and FullShoppingCartFormHandler help with?

Answer:

All parts of the checkout process including add to cart, shipping, billing, and committing.

Section 2 
Check Your Understanding

What form handler can be used to create credit card payment groups?

Answer:

CreateCreditCardFormHandler and UpdateCreditCardFormHandler

Section 2
Check Your Understanding

What droplet is used for recalculating pricing information?

Answer:

RepriceOrderDroplet.

Section 2
Check Your Understanding

What happens to the order when you use a SaveOrderFormHandler?

Answer:

It is added to the list of user's saved orders. A new blank one is created.

Summary

- The order checkout process can vary depending on the requirements and complexities involved.
- ATG supports: express checkout and complex checkout.
- ShippingGroupDroplet and ShippingGroup FormHandler are used to associate shipping groups with the order.
- PaymentGroupDroplet and PaymentGroup FormHandler are used to associate payment groups with the order.
- Form handlers are provided to create and update shipping and payment groups.
- CommitOrder FormHandler is used to commit an order to the system.

Q&A

ORACLE IS THE INFORMATION COMPANY


ORACLE®