Layers: -

In “SAP Comm” – We have structured way of writing the code.

You take any application – there will be **2 flows**: -

1st Flow = Registration (You Enter the data in the Form)

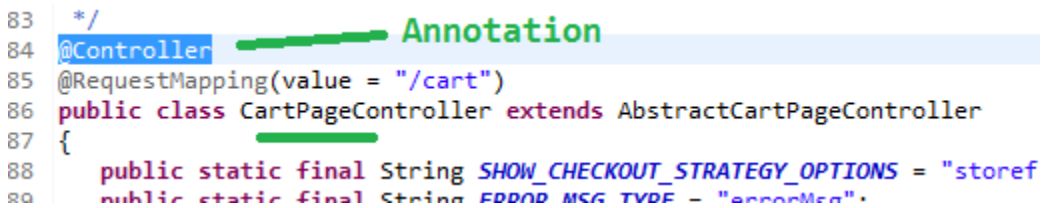
= **Uni–Directional** Flow

PL (JSP file / Tag file / HTML file / JS file / CSS file / ….) → Controller → Façade Layer (FL) → Service Layer (SL) → Perform the DB Save Operation.

Controller = Java file

Q: When can you say a Java file = Controller?

@Controller (Annotation)



1 Controller can call N number of other controllers.

FL = Java file.

1 FL can call another FL. So, we can have N number of FL.

SL = Java file.

1 SL can call another SL. So, we can have N number of SL.

We can have 1 / more JSP / Tag files for the same.

Q: When to write code in JSP File & When to write code in Tag file?

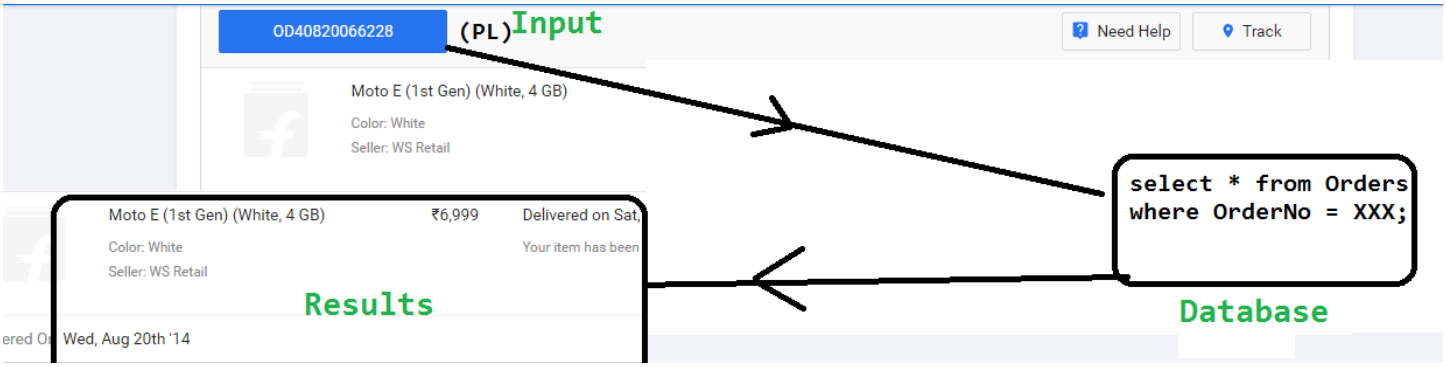
Let's say -- U have XXX code & you want to re-use that code in more than 1 place then put that XXX code in Tag file.

Else -- You can create JSP File

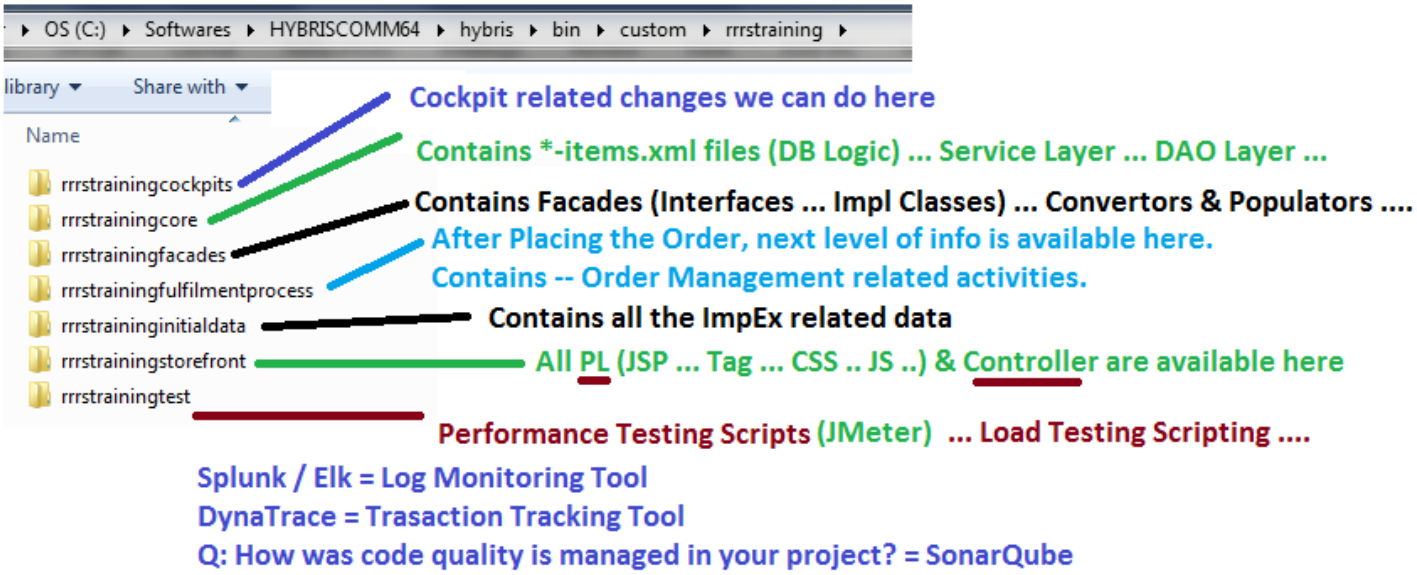
2nd Flow = Order Status (Enter Order No & Display the status)

= **Bi–Directional** Flow

PL (JSP file / Tag file / HTML file / JS file / CSS file / ….) → Controller → Façade Layer (FL) → Service Layer (SL) → DAO Layer (You Write the Flexible Search Query) → Service Layer → Façade Layer → Controller → Presentation Layer (PL) & Display the results.



Note: - In "SAP Comm" projects, we will be having 7 (or) More Exts.



Note: - "SAP Comm" is having different layers for coding.

Q = Can I write all the code in 1 file? = Yes, you can.

Contact Us = ChennaReddyTraining@RRRS.CO.IN

Q = Why do we need to write coding in those many layers?

= **Loosely Coupling**.

DAO Layer (Here you write the Flexible Search Queries = FSQ)? ➔

If you take any DB, what can we do? =

1) **DDL** = Create table … Create Cols ……

For DDL Operation, in "SAP Comm" what do we have? = ***-items.xml**

2) Insert Record … Update Record … Delete Record

For this DML Operations, in "SAP Comm" what do we have? = **ImpEx**

3) Get the Records (Select Command)

For this DML operation, in "SAP Comm" what do we have? = FSQ

Oracle – Select == **"SAP Comm" – FSQ**

Note: - In "SAP Comm" – "80 - 90%" of the code already available. Only "20 - 10%" of code we write based on customers need.

Now – It's import for us to find out existing code & write the new code on top of it.

Note: - In eCom Site – We will be having different pages. Example: -

1) Cart Page 2) Checkout Page

3) Quick Order Page 4) Product List Page 5) ….

For every page there will be 1 Controller.

For Cart Page → We have "**CartPageController.java**"

For Checkout Page → We have "**MultiStepCheckoutController.java**"

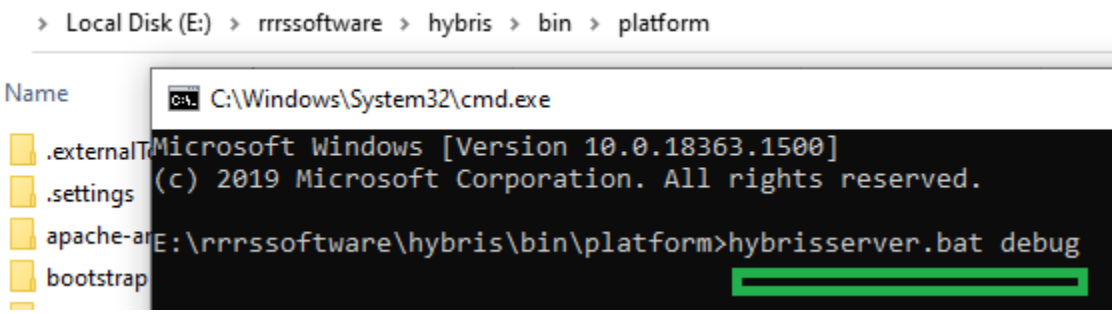
For Quick Order Page → We have "**QuickOrderPageController.java**"

Q = How to identify right files? = Put debug in code & find out.

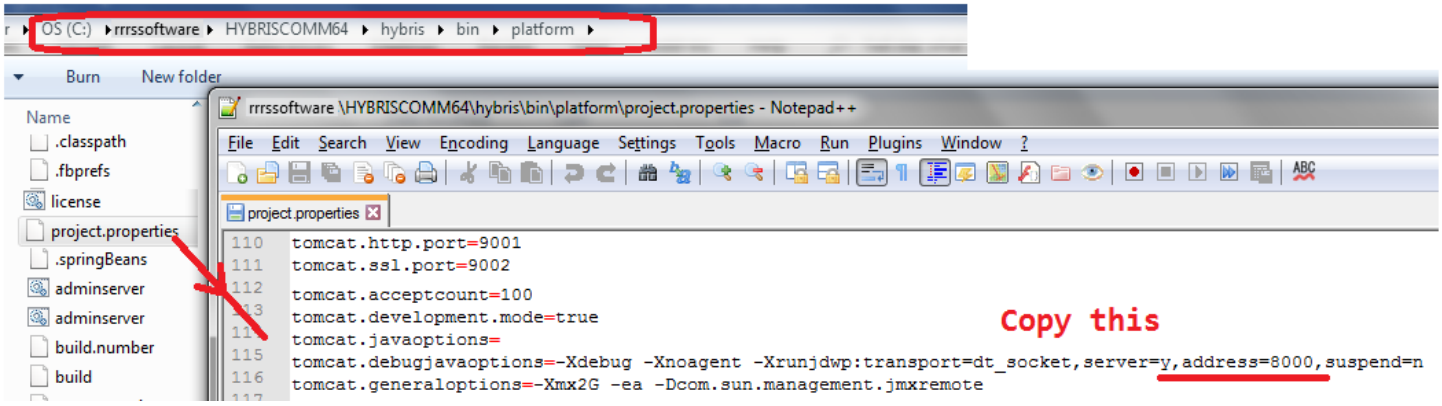
Q = How to debug the code? =

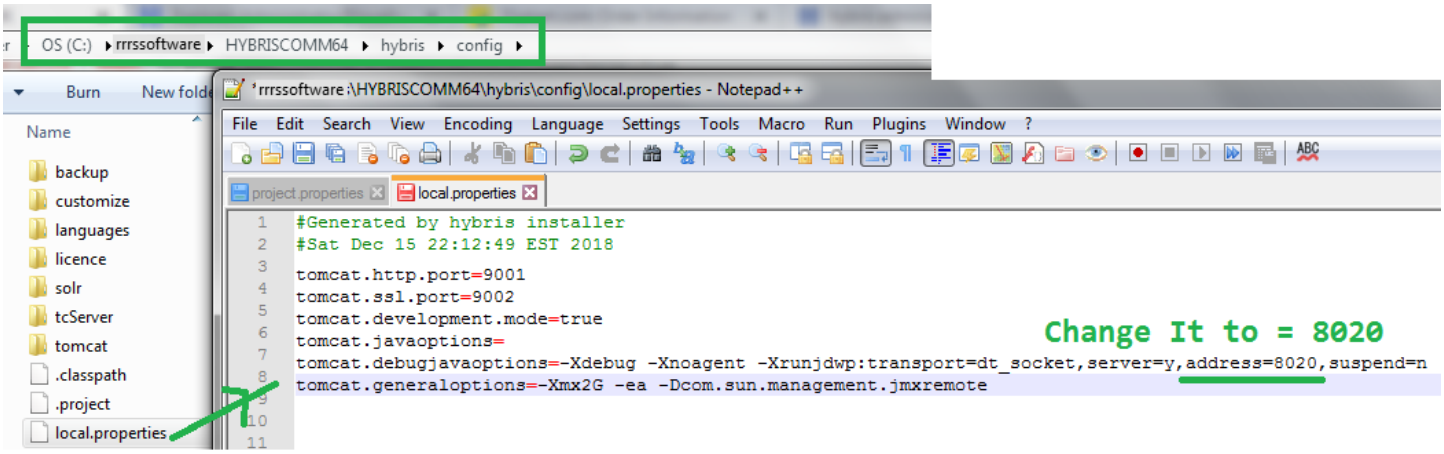
Step 1 = Start the server in debug mode

C:\rrrssoftwares\hybris\bin\platform>**hybrisserver.bat debug**



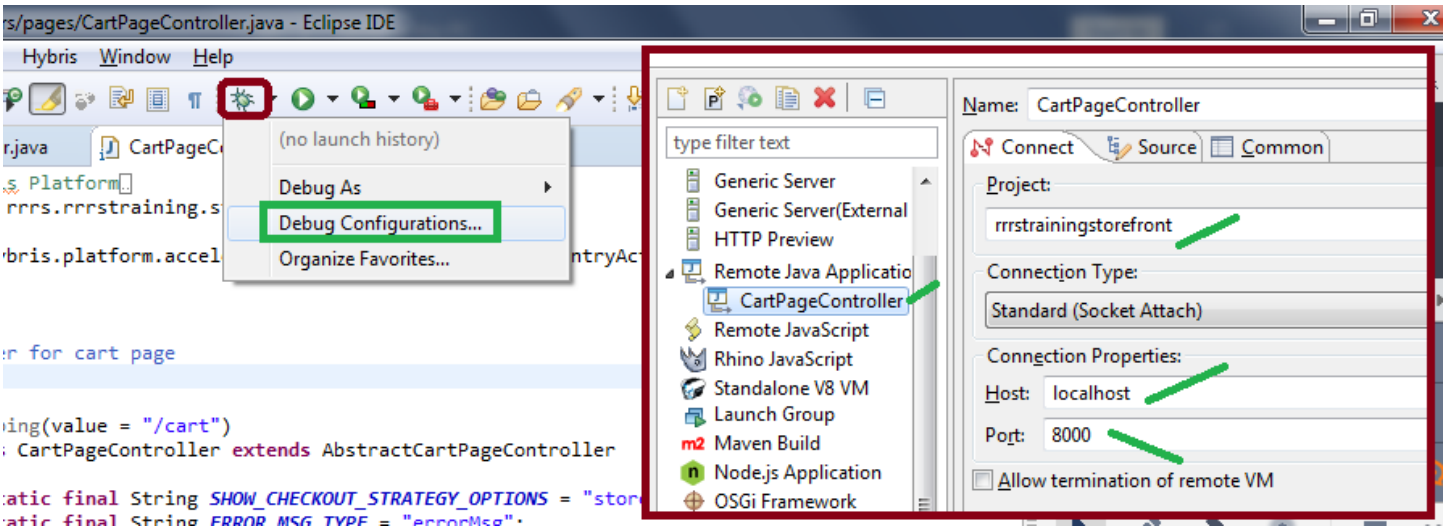
Note: - debug runs at specific port (=**8000 default**). Sometimes, you get error saying that port is already used. So – How to change the port / How find the current port?





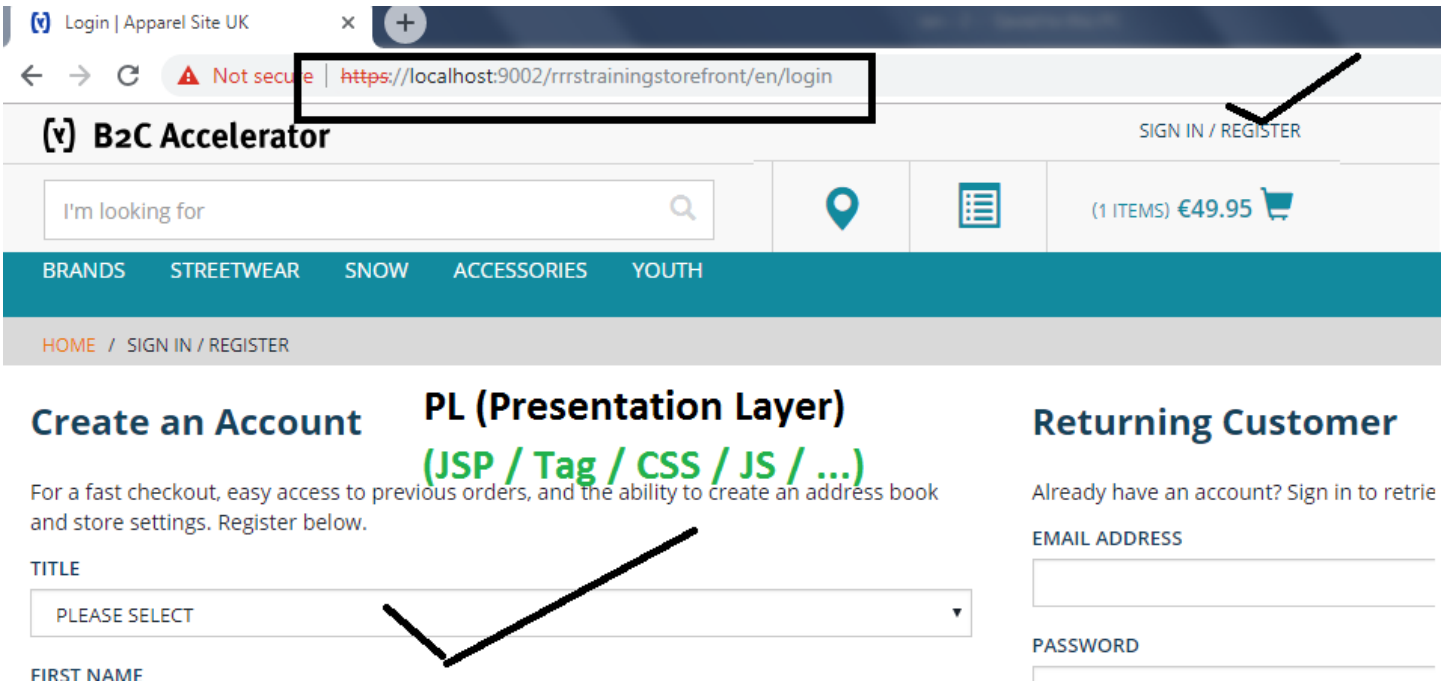
Contact Us = ChennaReddyTraining@RRRS.CO.IN

Step 2 = In Eclipse, Configure the debug settings



Local everything is working fine...
Dev System also everything is fine...
But UAT System Cart page is not working...
Sol1 = Logs
Sol2 = Start UAT in Debug mode
In Eclipse -- Host = UAT ...
Port = UAT ...
Start Debug

1st Flow = Registration Example: -



PL (Presentation Layer)

(JSP / Tag / CSS / JS / ...)

Contact Us = ChennaReddyTraining@RRRS.CO.IN

Req1 = How to add another filed called "**Enter Address**" after 1st Name & Last Name?

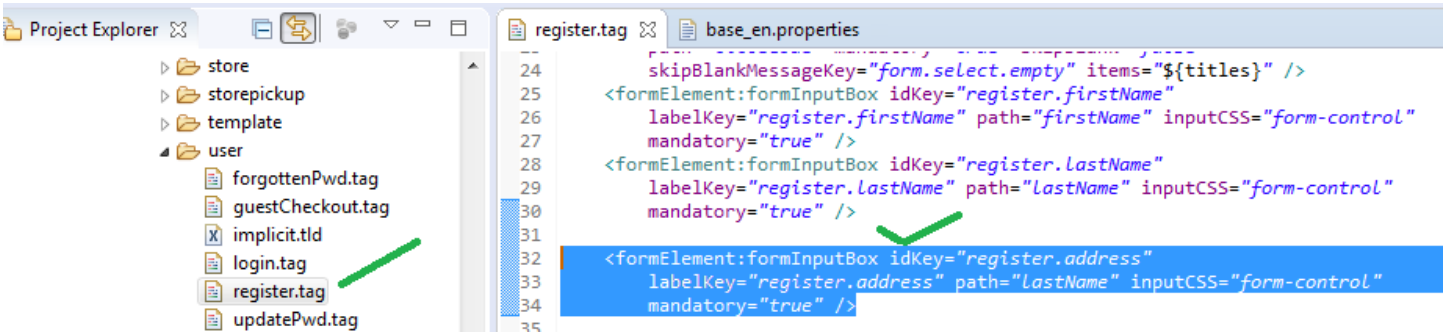
Step 1 = Q - How to find existing PL layer?

Sol1 = We know that, PL logic is available in **"*storefront"** (*=RRRSTraining) Ext.

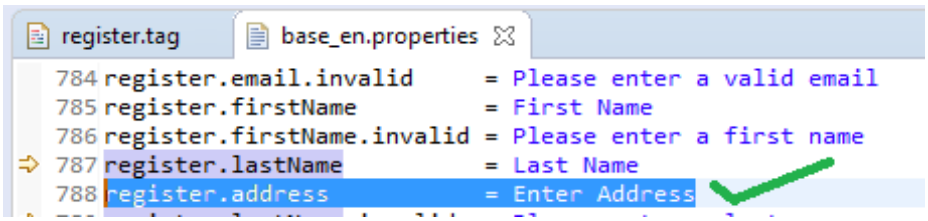
Note: - Hybris 6.X version on words – It's responsive.

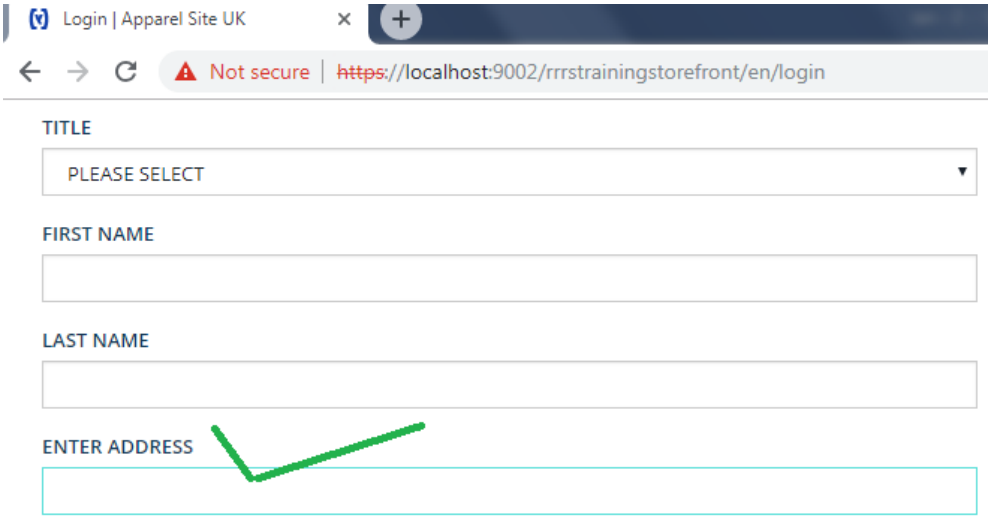
So, go to ***storefront** Ext & find the corresponding file & do the changes.

/*storefront/web/webroot/WEBINF/tags/responsive/user/**register.tag**



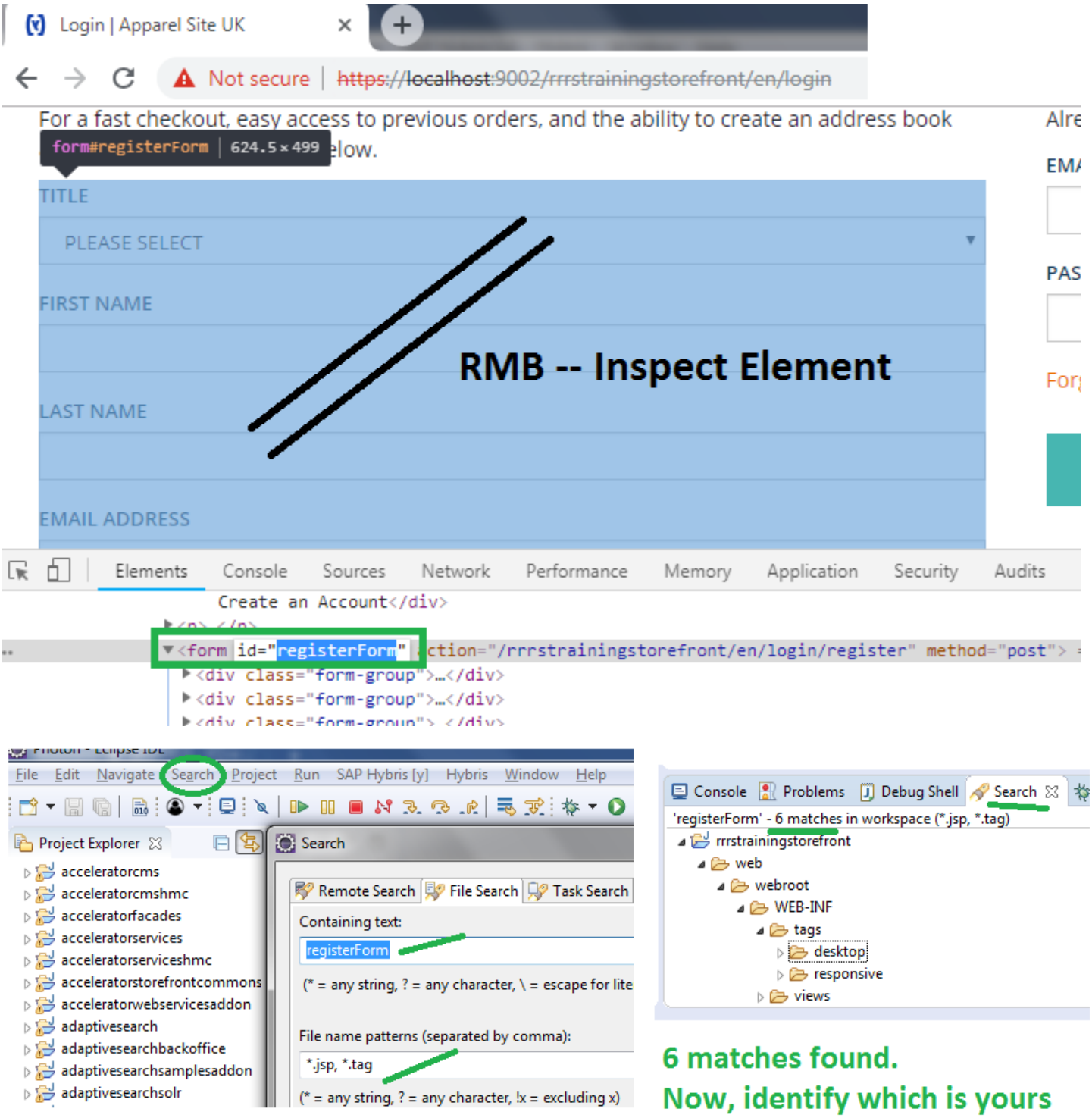
/*storefront/web/webroot/WEBINF/messages/**base_en.properties**





Contact Us = ChennaReddyTraining@RRRS.CO.IN

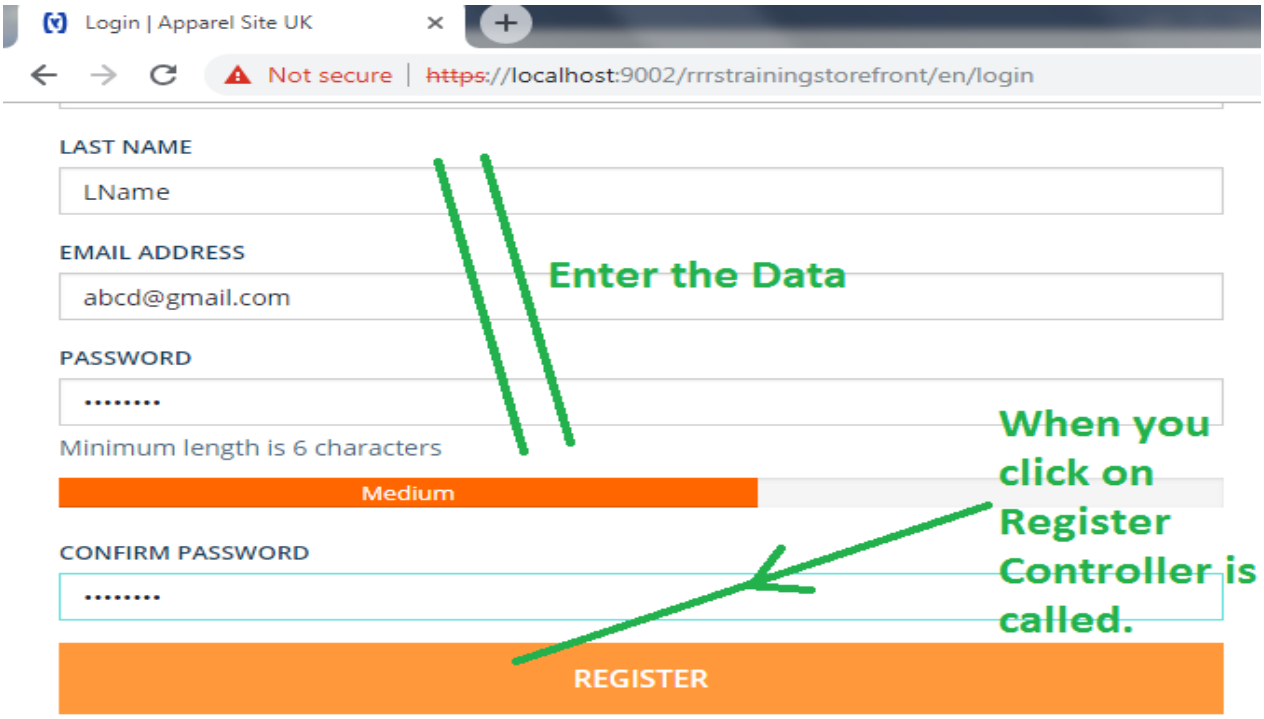
Sol2 = Go to the required place where you want the changes, RBM → Inspect Element, identify some unique element (ID name or Class or …)



From 6 matches, find which our current requirement file is. Open that file → Delete the code. Save it. Refresh the browser & see the results.

Contact Us = ChennaReddyTraining@RRRS.CO.IN

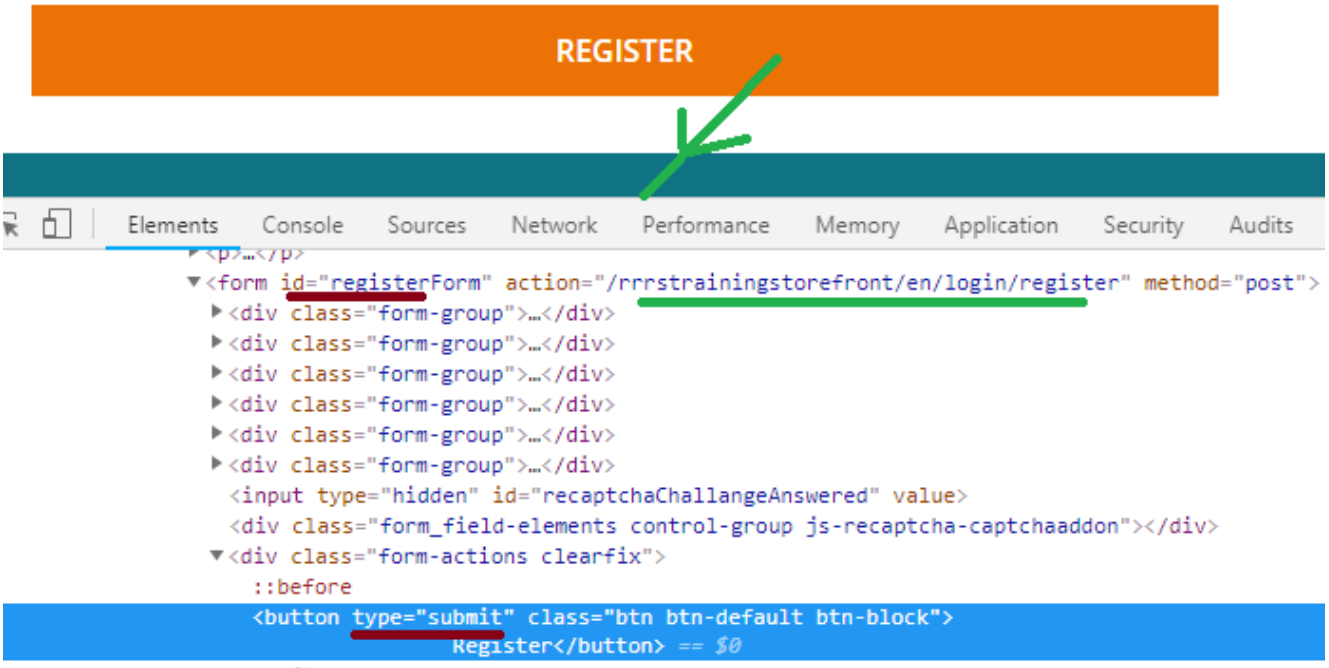
Q = What are we trying to do in PL? = We are entering lots of field values (FName, LName, Address and …)



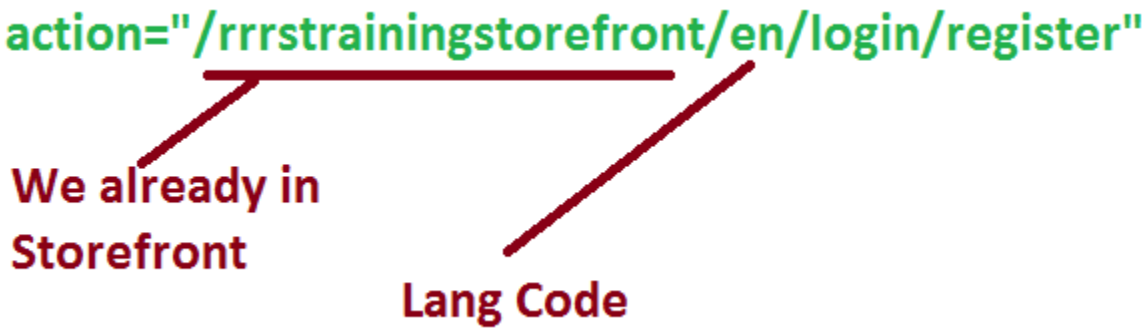
Note = After entering all the data → **Register**, **Controller** will be called.

Controller code / file will be available in ***storefront** Ext.

Step 2 = Q: How to find out corresponding Controller?

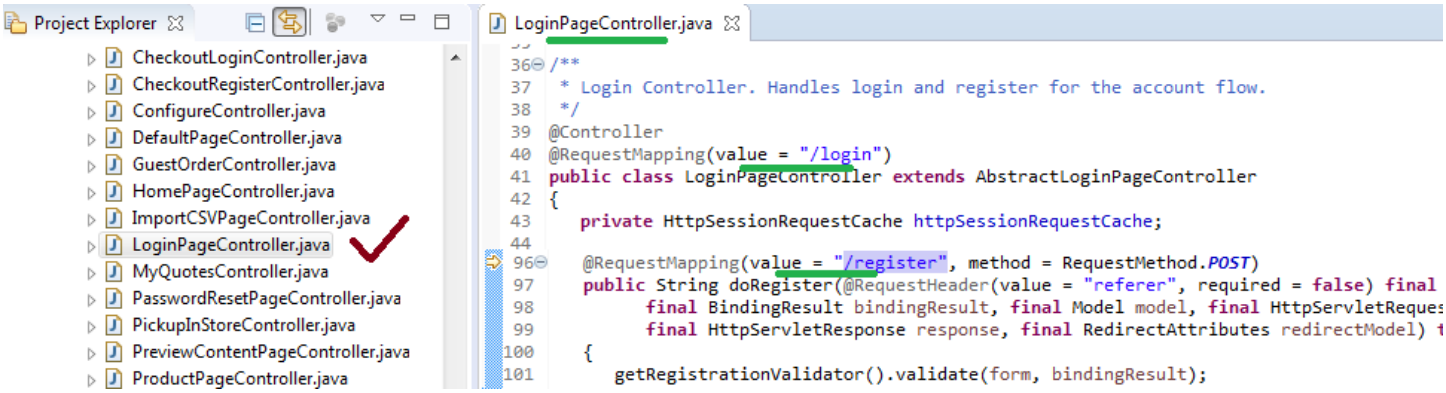


action="/**rrrstrainingstorefront**/en/login/**register**"



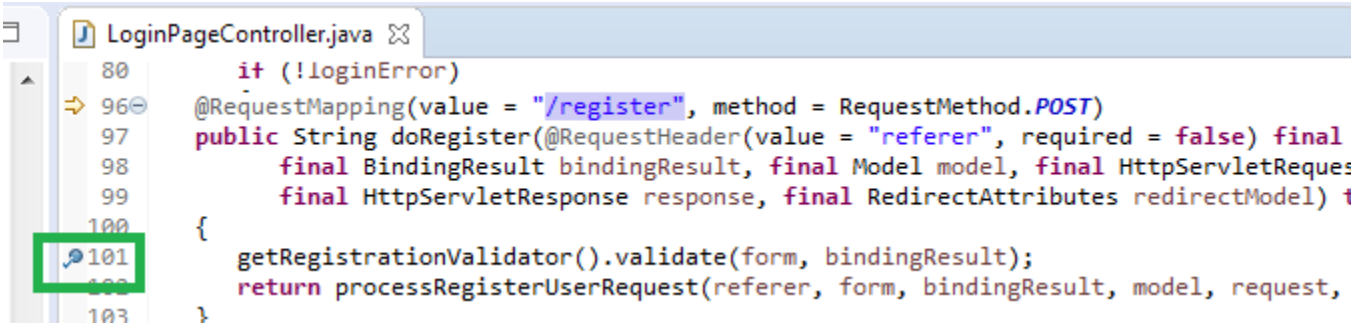
/register = This will be a request mapping at method level.

/login = This will be a request mapping at class level.



Q = What is the guarantee that above file is the Correct Controller?

Start the server debug mode & see the flow is coming in this file.



Note = When we click on Register button in **storefront**, if the flow coming here then it's the file & method.

```
96    @RequestMapping(value = "/register", method = RequestMethod.POST)
97    public String doRegister(@RequestHeader(value = "referer", required = false) final String referer, final RegisterForm form,
98          final BindingResult bindingResult, final Model model, final HttpServletRequest request,
99          final HttpServletResponse response, final RedirectAttributes redirectModel) throws CMSItemNotFoundException
100   {
101      getRegistrationValidator().validate(form, bindingResult);
102      return processRegisterUserRequest(referer, form, bindingResult, model, request, response, redirectModel);
103   }
```

```
@RequestMapping(value = "/register", method = RequestMethod.POST)
public String doRegister(@RequestHeader(value = "referer", required = false) final String referer, final RegisterForm form,
      final BindingResult bindingResult, final Model model, final Http
      final HttpServletResponse response, final RedirectAttributes re
{
   getRegistrationValidator().validate(form, bindingResult);
   return processRegisterUserRequest(referer, form, bindingResult, mo
}
}
```

Console | Problems | Debug Shell | Search | Debug | Servers | (x)= Variables

- form= RegisterForm (id=330)
  - captcha= null
  - checkPwd= "12341234" (id=335)
  - email= "abcd@gmail.com" (id=339)
  - firstName= "FName" (id=340)
  - lastName= "LName" (id=341)
  - mobileNumber= null
  - pwd= "12341234" (id=28682)
  - titleCode= "mr" (id=28683)

Note = Whatever we entered in UI layer is available in **RegisterForm** Obj.

Q = What exactly we are doing in controller?

Q1 = In PL, we are entering PWD & Confirm PWD. If these 2 are not same, then is it required for us to continue next steps? = No.

Q2 = In PL, we are entering email ID. If email ID does not have @ symbol, then is it required for us to continue next steps? = No.

====== In PL, we are entering lots of data. It's time to validate those.

Note: - For validation purpose – We have **Validator FWK**.

Validator FWK is having **validate ()** method.

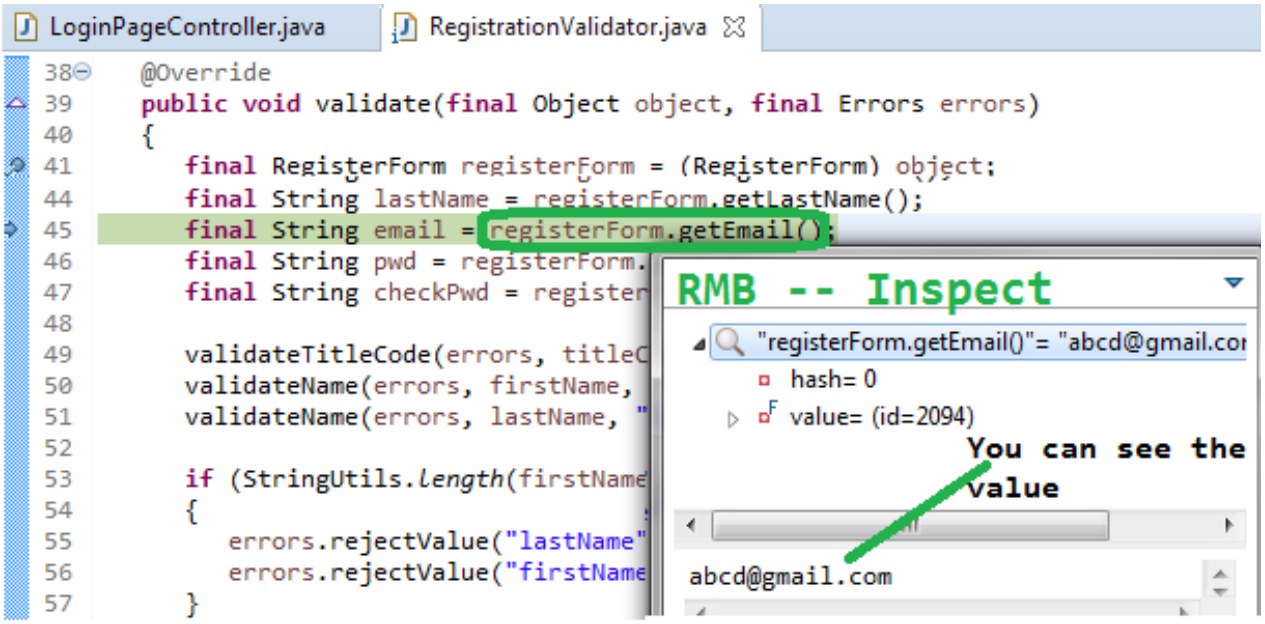
We need to override validate () method.

```
96    @RequestMapping(value = "/register", method = RequestMethod.POST)
97    public String doRegister(@RequestHeader(value = "referer", required = false)
98          final BindingResult bindingResult, final Model model, final HttpServle
99          final HttpServletResponse response, final RedirectAttributes redirectM
100   {
101      getRegistrationValidator().validate(form, bindingResult);
102      return processRegisterUserRequest(referer, form, bindingResult, model, re
103   }
```

LoginPageController.java | RegistrationValidator.java

```
38    @Override
39    public void validate(final Object object, final Errors errors)
40    {
41       final RegisterForm registerForm = (Re
42       final String titleCode = registerForm
43       final String firstName = registerForm
44       final String lastName = registerForm.
45       final String email = registerForm.get
46       final String pwd = registerForm.getPw
47       final String checkPwd = registerForm.
48
49       validateTitleCode(errors, titleCode);
50       validateName(errors, firstName, "firs
51       validateName(errors, lastName, "lastN
```

- object= RegisterForm (id=330)
  - captcha= null
  - checkPwd= "12341234" (id=335)
  - email= "abcd@gmail.com" (id=339)
  - firstName= "FName" (id=340)
  - lastName= "LName" (id=341)
  - mobileNumber= null
  - pwd= "12341234" (id=28682)
  - titleCode= "mr" (id=28683)



Note = After validation is successful – Do below: -

Q3 = In PL, assume that you entered the **Confirm PWD**.

Are we really storing the Confirm PWD DB? = No.

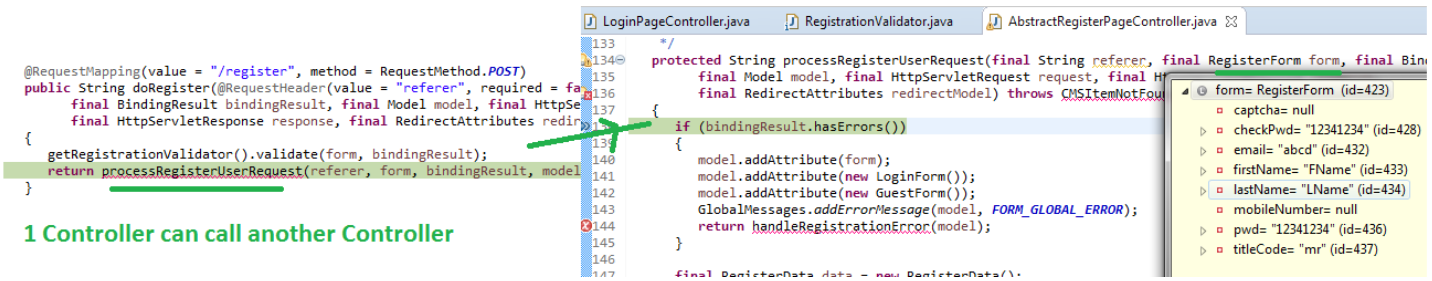
Q4 = In PL, assume that you entered the **captcha**.

Are we storing the captcha in DB? = No.

====== In PL, we are entering lots of data & we may not store all the data in DB. So, it is not required to carry the data to next steps if not stored in DB.

So, create another Obj & put required into in this.

This Obj is called "**Data Obj**".



Once Data Obj is ready then call the **Façade Layer**.

After Data Obj is ready = Call the FL = `getCustomerFacade().register(data);`

Q1 = What is the input for Controller?

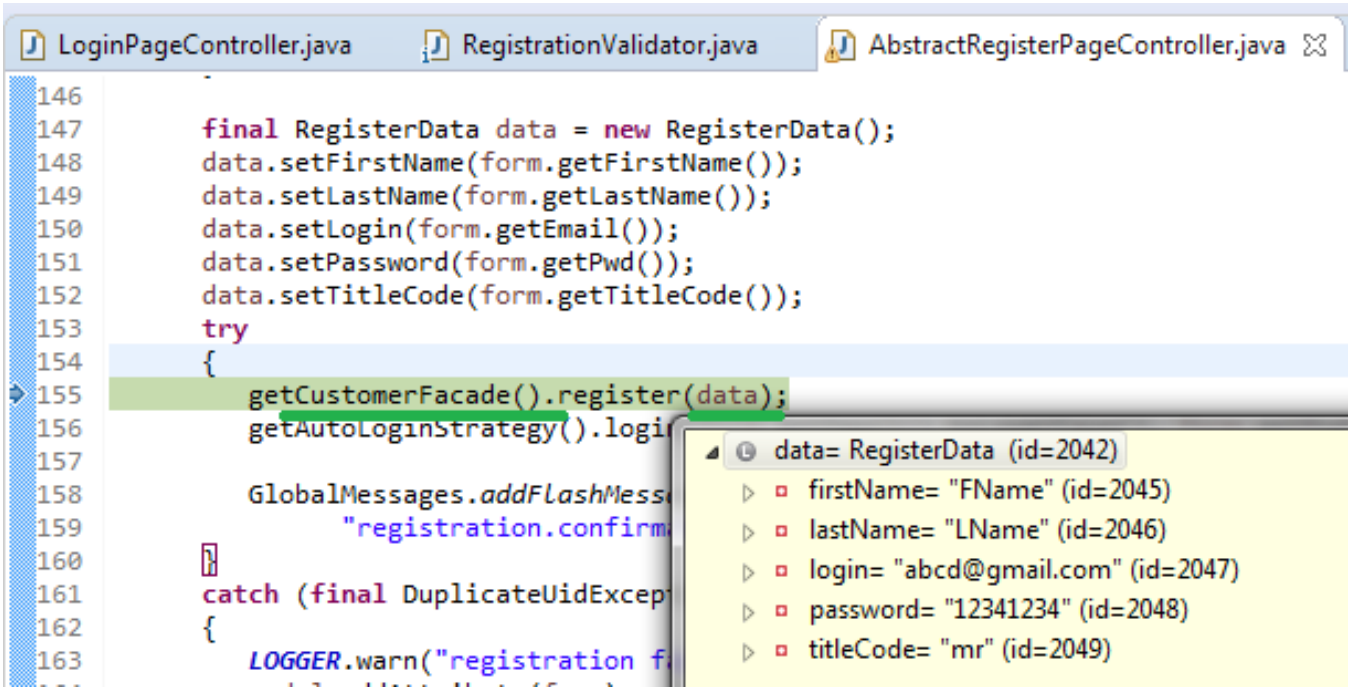
= Form Obj

Q2 = What we did in Controller? =

1) Validations
2) Creating Data Obj

Q3 = What is the output of the Controller?

= Data Obj & Call Façade Layer.



Q = How to call the Façade Layer?

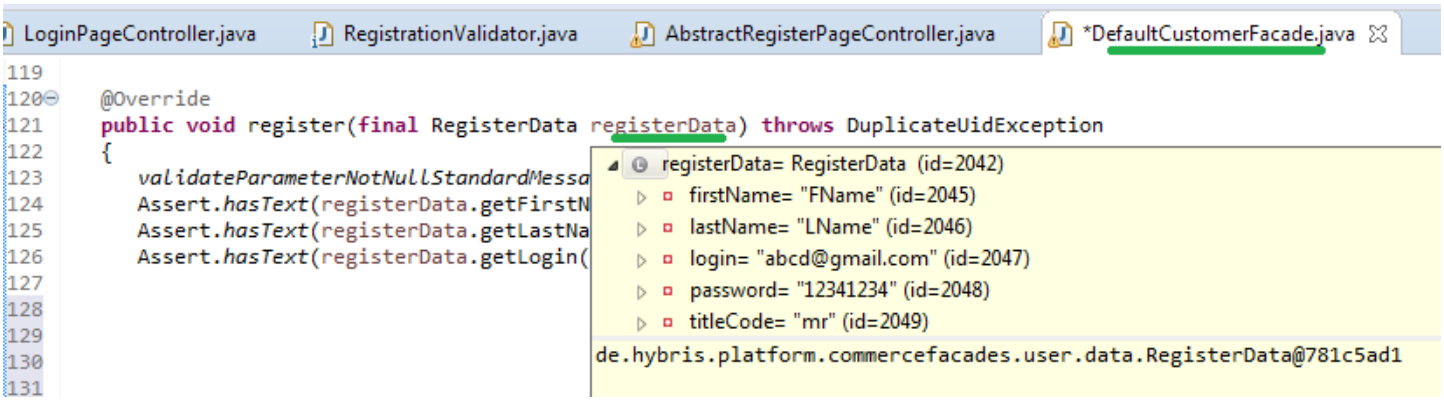
= get**Customer**Facade().register(data);

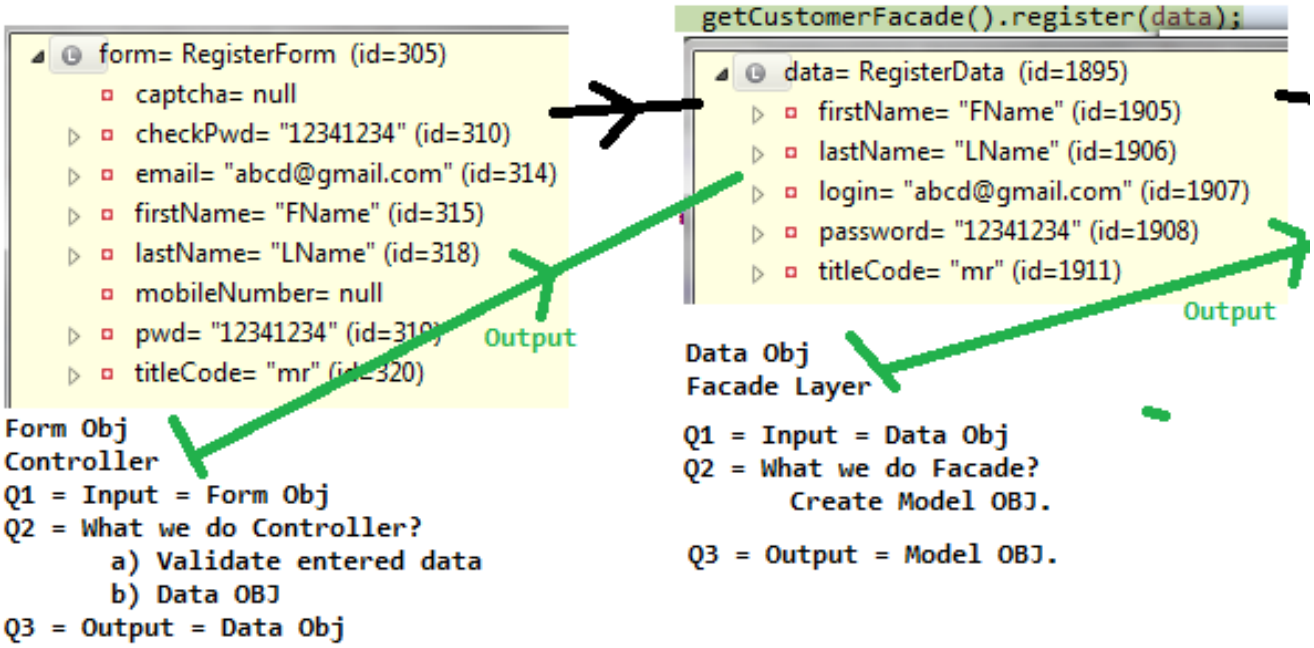
General Syntax =

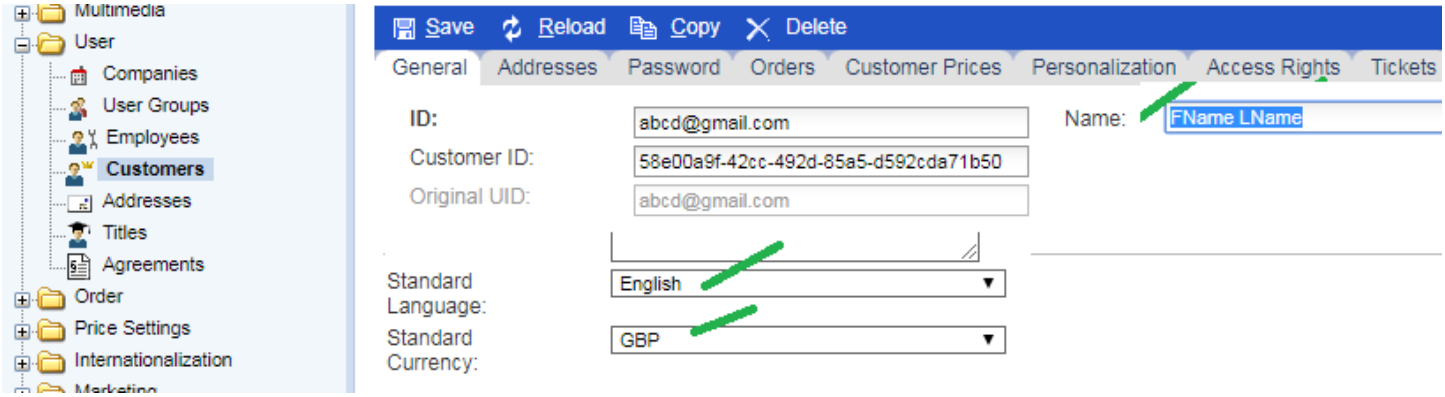
get**XXX**Facade().method(==);

XXX = Customer / Order / Cart / ….

Step 3 = Façade Layer: -







Q5 = In **Data Obj**, we have First Name & Last Name. But in DB, we only have Name & it's concatenation of 1st name & last name.

Q6 = In Data Obj, we don't have Lang … Currency …. But in DB, we have Lang & Currency and ….

===> That means, Ur Data Obj is not TRUE representation of DB table customer record.

Now, it's time to create another Obj called "**Model Obj**", which will be TRUE representation of your DB table customer record.

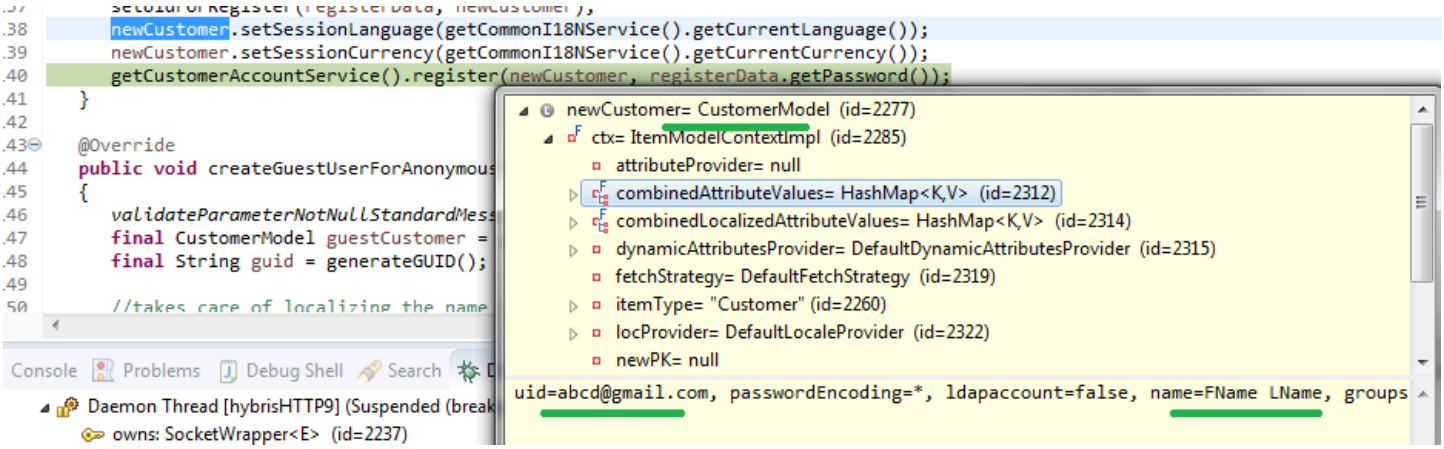
For **Customer** → We have **CustomerModel**

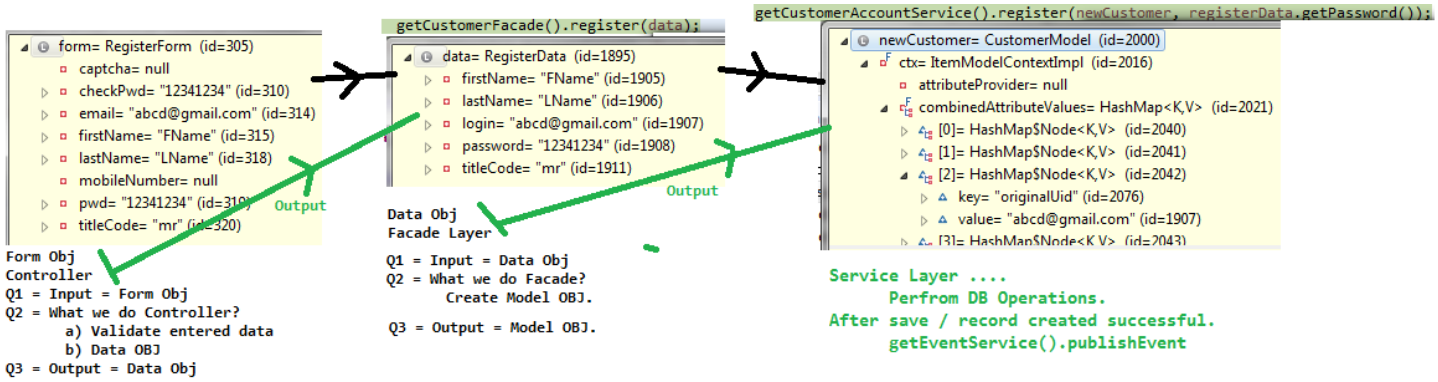
For **Order** → We have **OrderModel**

➔ General Syntax == For **XXX** item type … We will have **XXX**Model

XXX = Customer / Order / Cart / Address / ….

Note: - Once Model Obj is ready then call the **Service Layer**.





Q1 = What is the input for Façade Layer? = Data Obj

Q2 = What we did in Façade Layer? = Creating the Model Obj

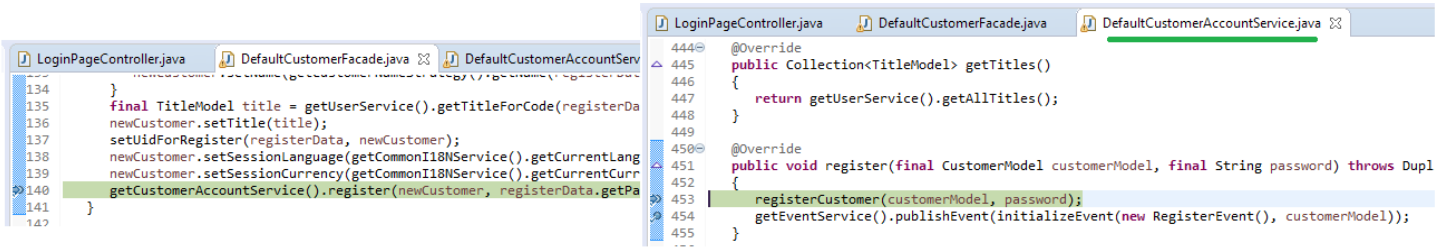
Q3 = What is Façade Layer output?

= Model Obj & also calling Service Layer.

Q4 = How to call the Service Layer? =

getCustomerAccountService().register(newCustomer, registerData.getPassword());

Step 4 = Service Layer: -

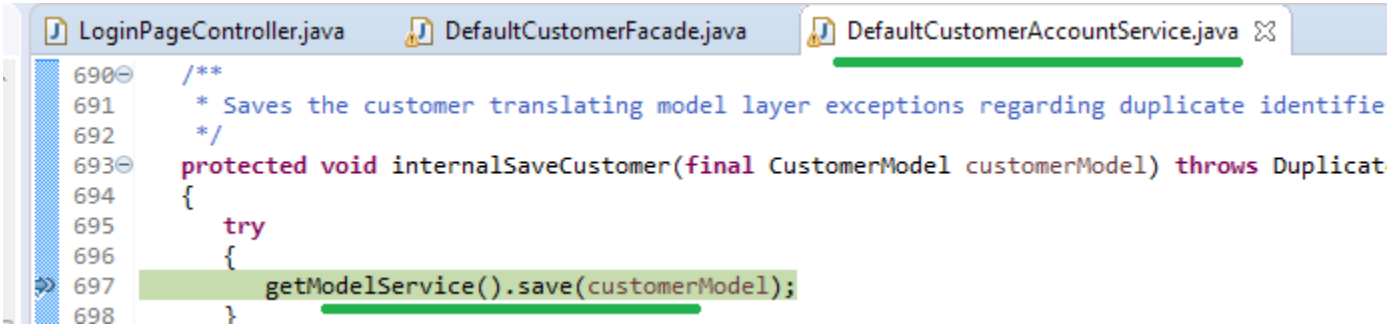


General Syntax: -

For Cart → We have *Cart*Service.java

For Customer → We have *Customer*Service.java

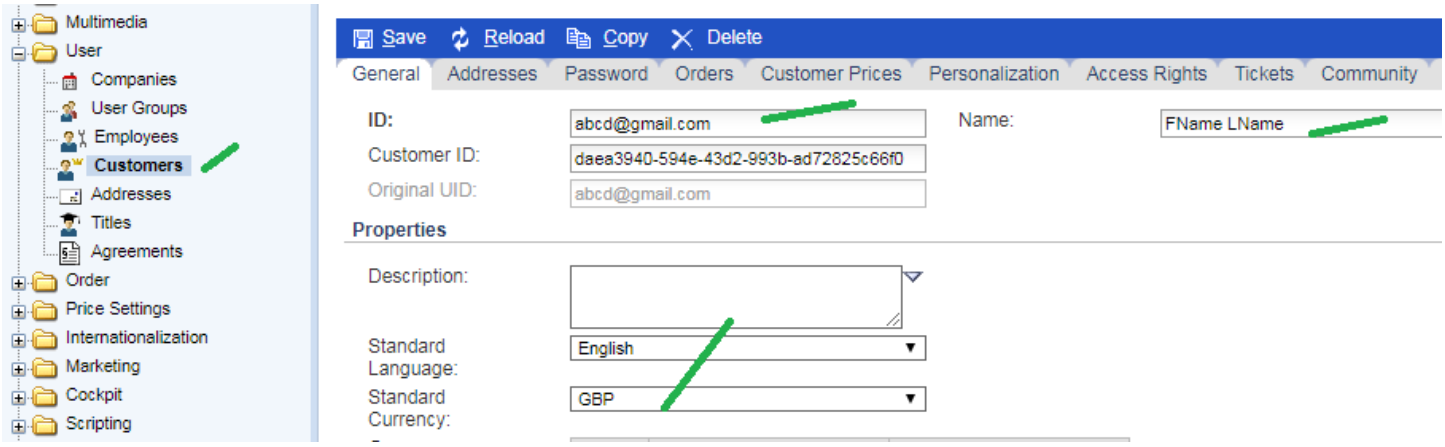
For Order → We have *Order*Service.java



Q = What is the general Syntax for creating the Record?

getModelService().save(**XXX**Model);

XXX = Customer / Cart / Address / Order / ….



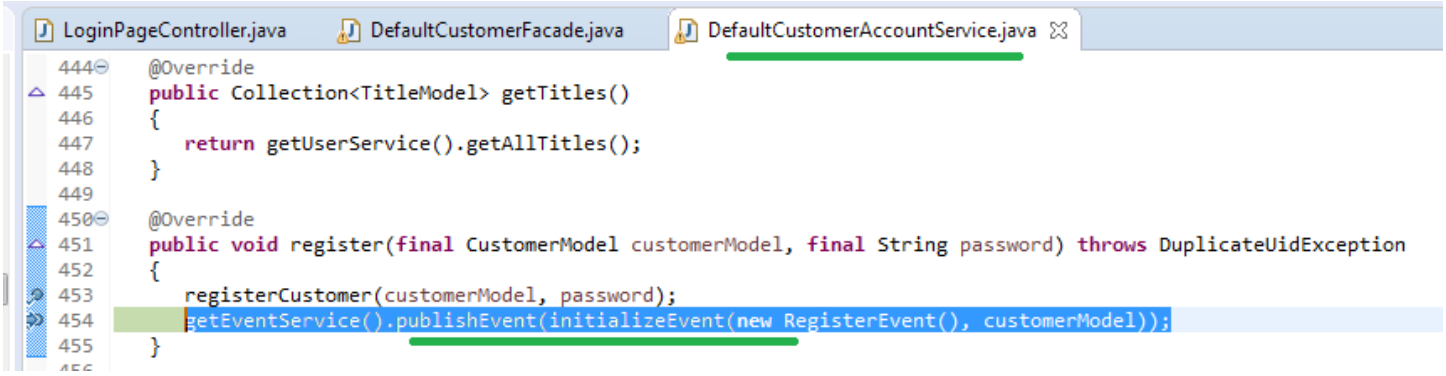
Q = What happen if getModelService().save() – Fails to perform the operation?

ModelSavingException

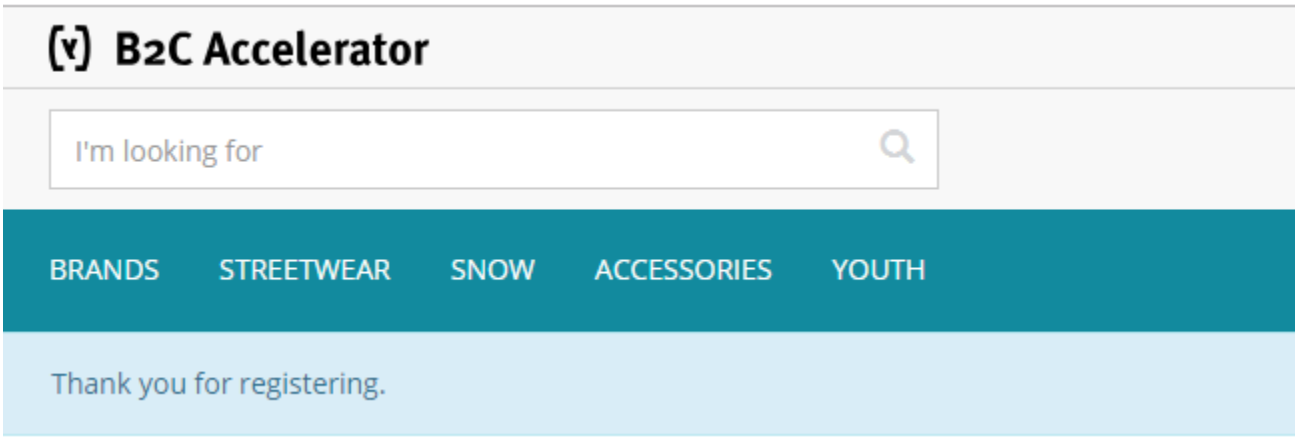
Q = What happen if getModelService().save() – Success to perform the operation?

Record will be inserted in the DB.

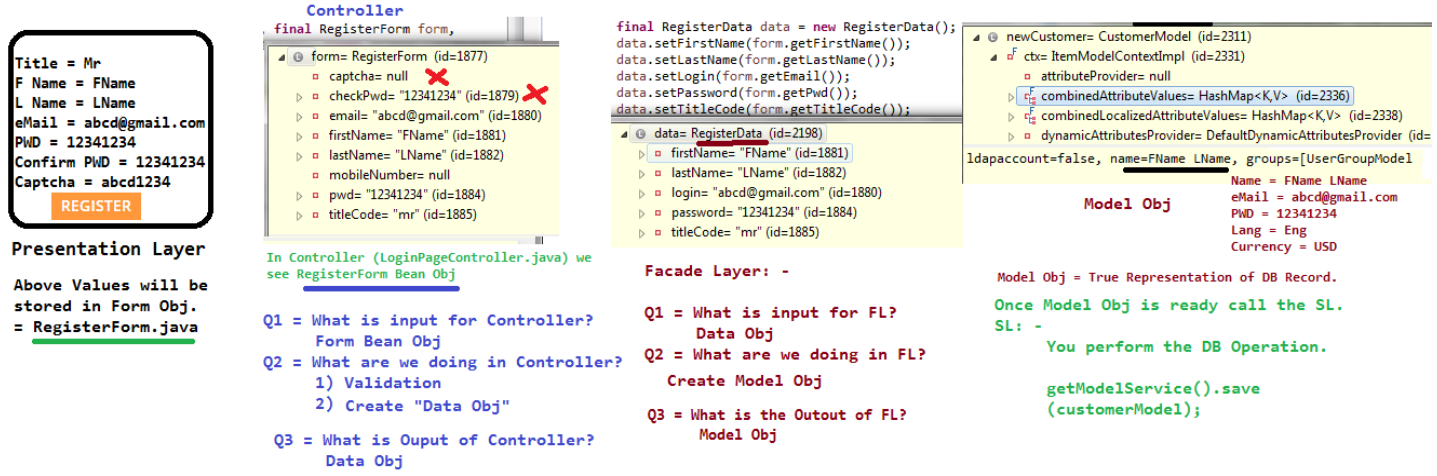
An event will be published stating that – Record insert is successful.



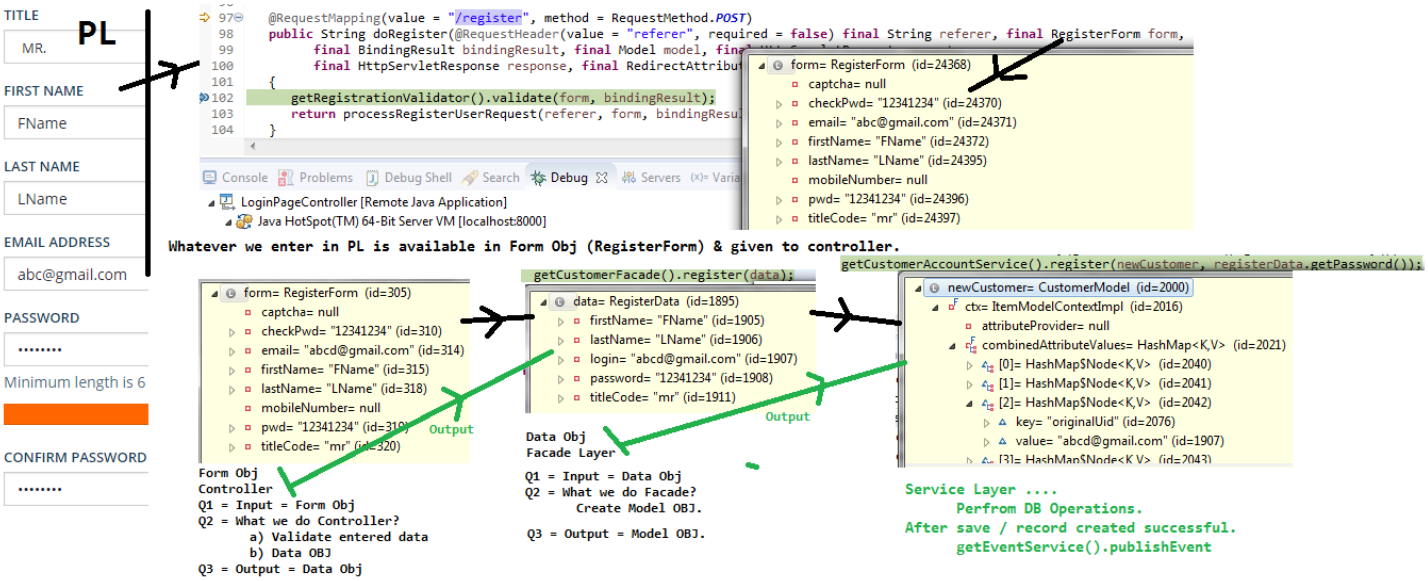
Results in Storefront: -



Overral Flow:-



Data Obj == Used for Storefront Purpose.
Model Obj == Used for DB operation Purpose.

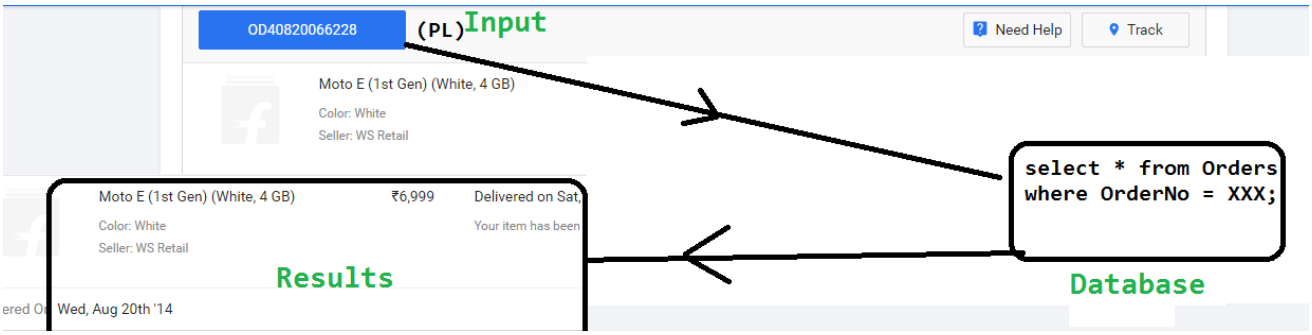


Contact Us = ChennaReddyTraining@RRRS.CO.IN

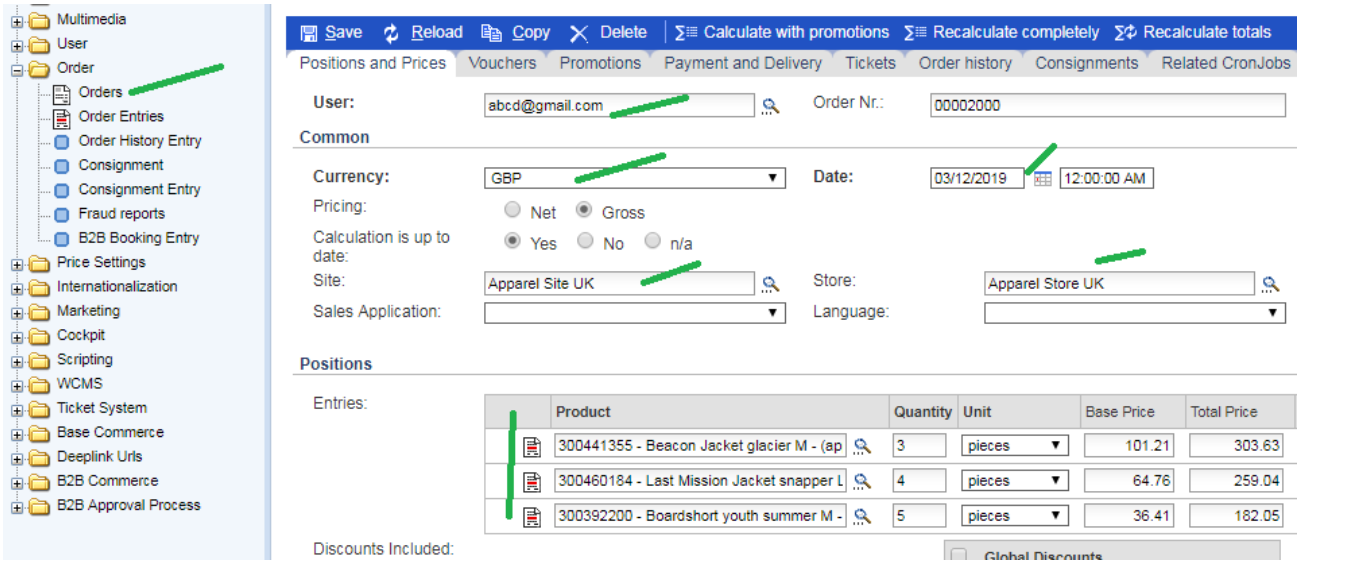
2nd Flow = Order Status (Enter Order No & Display the status)

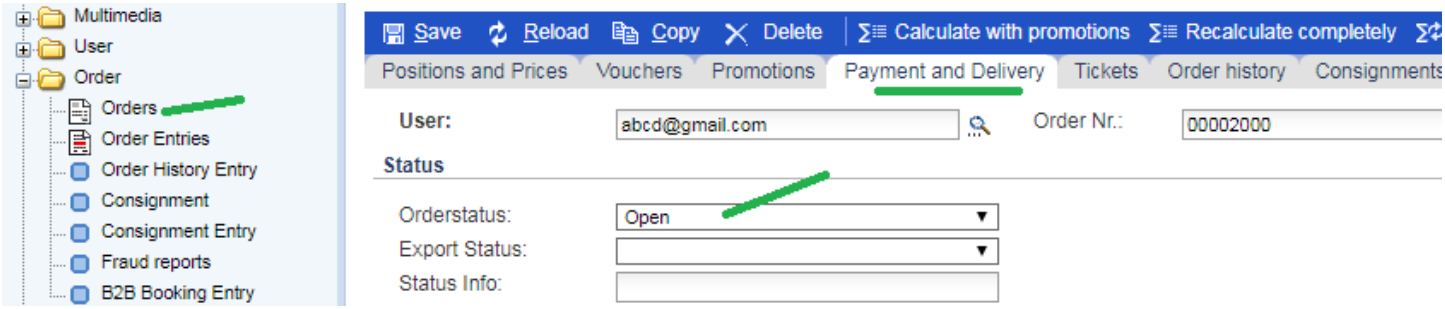
= **Bi–Directional Flow**

PL (JSP file / Tag file / HTML file / JS file / CSS file / ….) → Controller → Façade Layer (FL) → Service Layer (SL) → DAO Layer (You Write Flexible Search Query) → Service Layer → Façade Layer → Controller → Presentation Layer (PL) & Display the results.

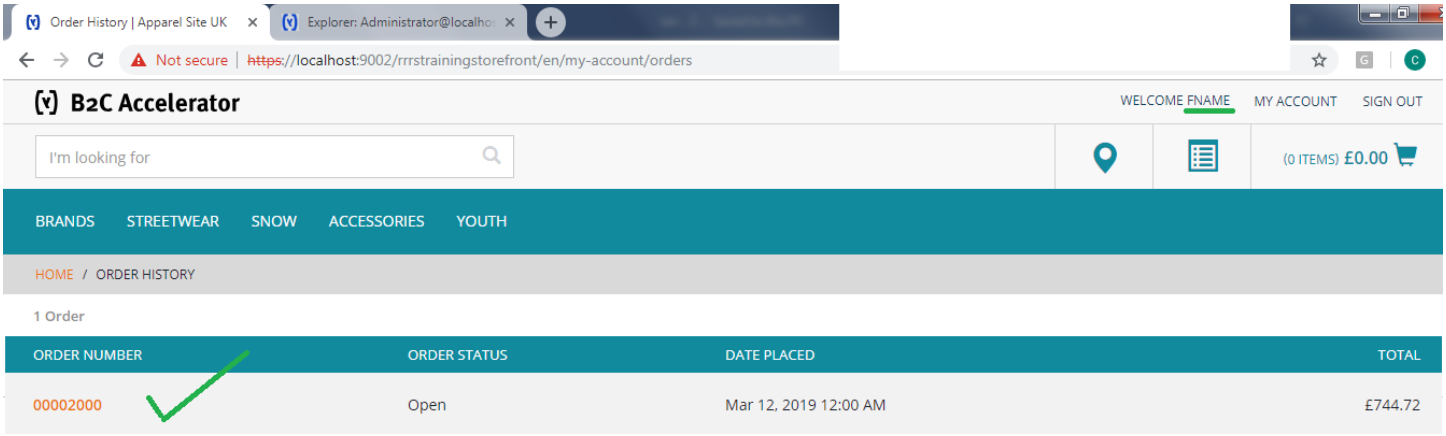


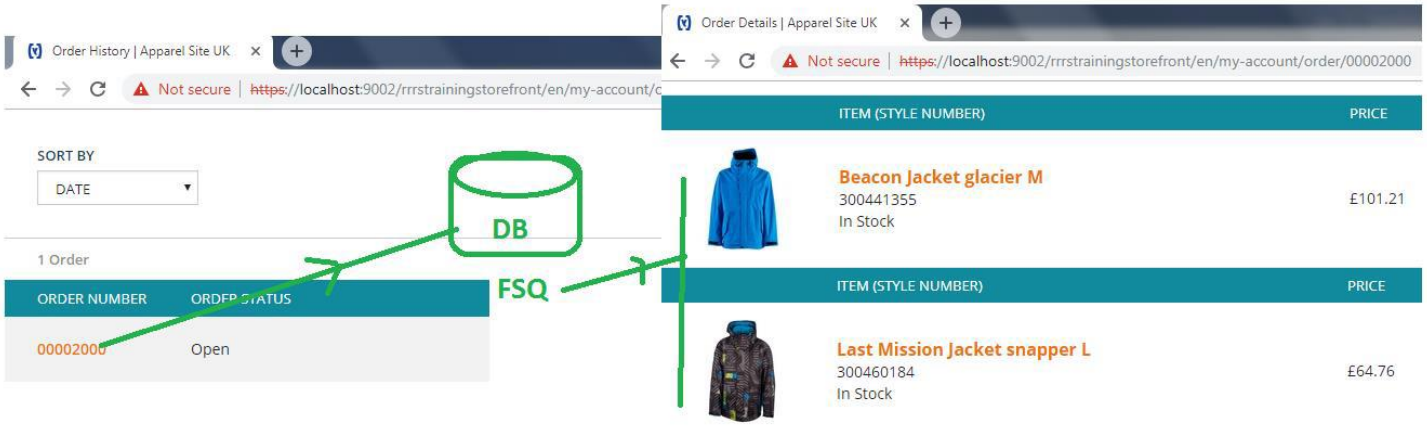
Q: How to place the Order from Backoffice / hmc?



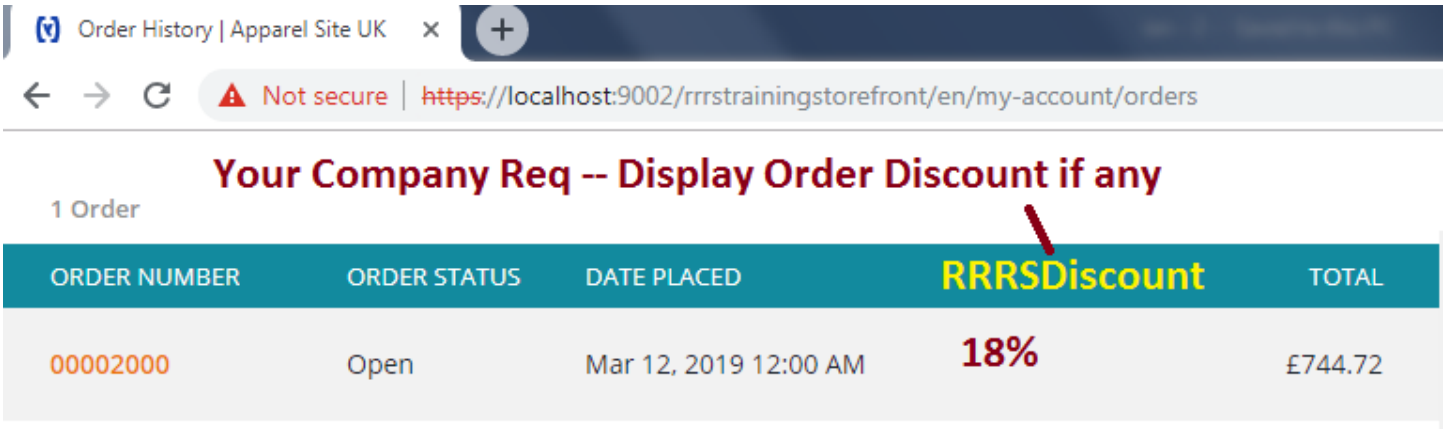


Contact Us = ChennaReddyTraining@RRRS.CO.IN





Step 1 = How to identify the PL?



Contact Us = ChennaReddyTraining@RRRS.CO.IN

base_en.properties | accountOrderHistoryPage.jsp

```
30          <nav:pagination top="true" msgKey="text.account.orderHistory.page" showCurrentPageInfo="tr
31      </div>
32      <div class="account-overview-table">
33          <table class="orderhistory-list-table responsive-table">
34              <tr class="account-orderhistory-table-head responsive-table-head hidden-xs">
35                  <th><spring:theme code="text.account.orderHistory.orderNumber" /></th>
36                  <th><spring:theme code="text.account.orderHistory.orderStatus"/></th>
37                  <th><spring:theme code="text.account.orderHistory.datePlaced"/></th>
38                  <th><spring:theme code="text.account.orderHistory.RRRSDiscount"/></th>
39                  <th><spring:theme code="text.account.orderHistory.total"/></th>
40              </tr>
41              <c:forEach items="${searchPageData.results}" var="order">
42                  <tr class="responsive-table-item">
```

base_en.properties | accountOrderHistoryPage.jsp

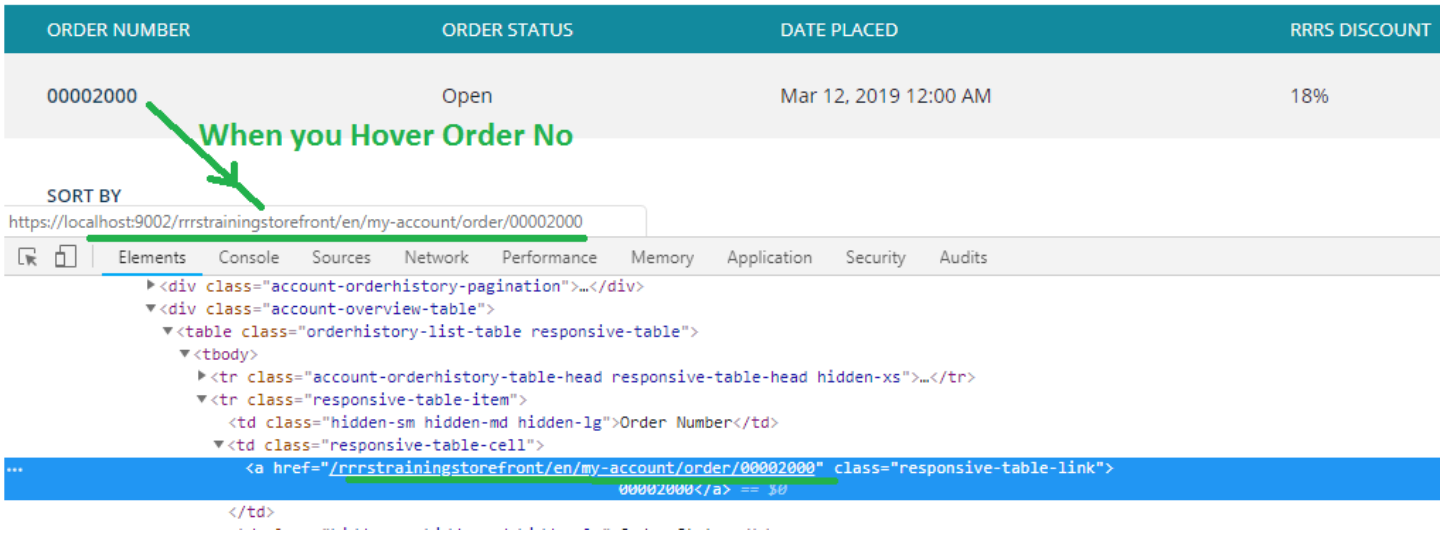
```
1020 text.account.order.qty                                  = Qty
1021 text.account.order.warning.storePickUpItems            = Reminder - Please
1022 text.account.order.yourOrder                           = Your Order
1023 text.account.order.totalSavings                        = You saved {0} on
1024 text.account.order.back.btn                              = Back
1025 text.account.orderHistory                              = Order History
1026 text.account.orderHistory.back                         = Back
1027 text.account.orderHistory.actions                      = Actions
1028 text.account.orderHistory.datePlaced                   = Date Placed
1029 text.account.orderHistory.RRRSDiscount                 = RRRS Discount
1030 text.account.orderHistory.mobile.page.currentPage      = {0} of {1}
```

base_en.properties | accountOrderHistoryPage.jsp | accountMyQuotesPage.jsp

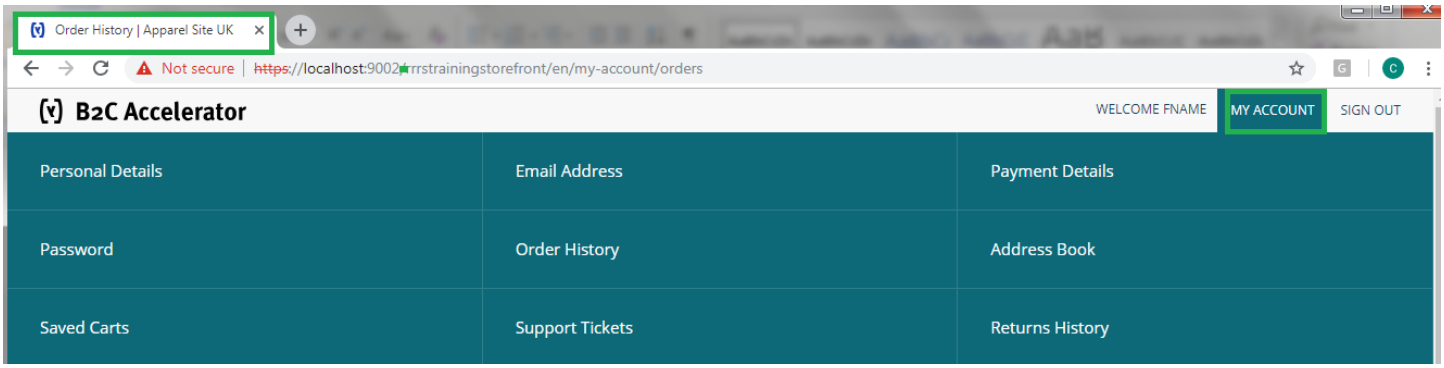
```
48                          </a>
49                      </td>
50                      <td class="hidden-sm hidden-md hidden-lg"><spring:theme code="text.account.orderHistory.orderStatus"/><
51                      <td class="status">
52                          <spring:theme code="text.account.order.status.display.${order.statusDisplay}"/>
53                      </td>
54                      <td class="hidden-sm hidden-md hidden-lg"><spring:theme code="text.account.orderHistory.datePlaced"/></
55                      <td class="responsive-table-cell">
56                          <fmt:formatDate value="${order.placed}" dateStyle="medium" timeStyle="short" type="both"/>
57                      </td>
58
59                      <td class="responsive-table-cell"> 18%
60                      </td>
61
62                      <td class="hidden-sm hidden-md hidden-lg"><spring:theme code="text.account.orderHistory.total"/></td>
63                      <td class="responsive-table-cell responsive-table-cell-bold">
64                          ${fn:escapeXml(order.total.formattedValue)}
65                      </td>
66                  </ycommerce:testId>
```

Step 2 = Find out the Controller.

Once we click on the “Order No”, the corresponding controller will be called.



Q: What are the different OOTB functionality are there in “**My-Account**” page?



<a href="/rrrstrainingstorefront/en/**my-account**/**order**/**00002000**" ===>

/order/ = Find mapping at method level. & /**my-account** = Find mapping at class level.

```
AccountPageController.java
 2  * [y] hybris Platform
11  package com.rrrs.rrrstraining.storefront.controllers.pages;
12
13  import de.hybris.platform.acceleratorfacades.ordergridform.OrderGridFormFacade;
90  @Controller
91  @RequestMapping("/my-account")
92  public class AccountPageController extends AbstractSearchPageController
93  {
94     private static final String TEXT_ACCOUNT_ADDRESS_BOOK = "text.account.addressBook";
95     private static final String BREADCRUMBS_ATTR = "breadcrumbs";
306    @RequestMapping(value = "/order/" + ORDER_CODE_PATH_VARIABLE_PATTERN, method = RequestMethod.GET)
307    @RequireHardLogIn
308    public String order(@PathVariable("orderCode") final String orderCode, final Model model,
309          final RedirectAttributes redirectModel) throws CMSItemNotFoundException
310    {
311       try
312       {
313          final OrderData orderDetails = orderFacade.getOrderDetailsForCode(orderCode);
314          model.addAttribute("orderData", orderDetails);
```

In Controller – What did we do yesterday?

1) We did the validation

Q: Do we need the validation for current scenario? = No need.

Yesterday (**Registration**) – Means, User is entering the data. So, there is possibility of user can enter wrong data. Hence did the validation.

Today (**Order Status**) – Means, User just click on Order No (User is not entering the order no), so no need the validation.

2) We have created the Data Obj

Step 3 = Façade Layer: -

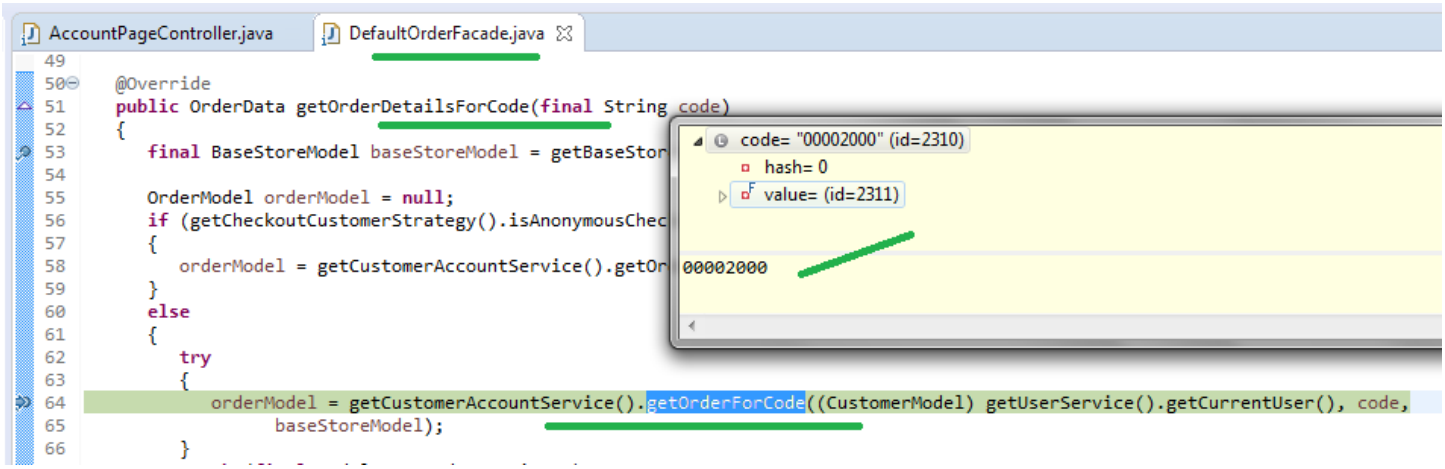
```
306    @RequestMapping(value = "/order/" + ORDER_CODE_PATH_VARIABLE_PATTERN, method = RequestMethod.GET)
307    @RequireHardLogIn
308    public String order(@PathVariable("orderCode") final String orderCode, final Model model,
309          final RedirectAttributes redirectModel) throws CMSItemNotFoundException
310    {
311       try
312       {
313          final OrderData orderDetails = orderFacade.getOrderDetailsForCode(orderCode);
314          model.addAttribute("orderData", orderDetails);
315
316          final List<Breadcrumb> breadcrumbs = accountBr
317          breadcrumbs.add(new Breadcrumb("/my-account/or
318                getI18nService().getCurrentLocale()), nu
```

orderCode= "00002000" (id=659)
hash= 0
value= (id=665)

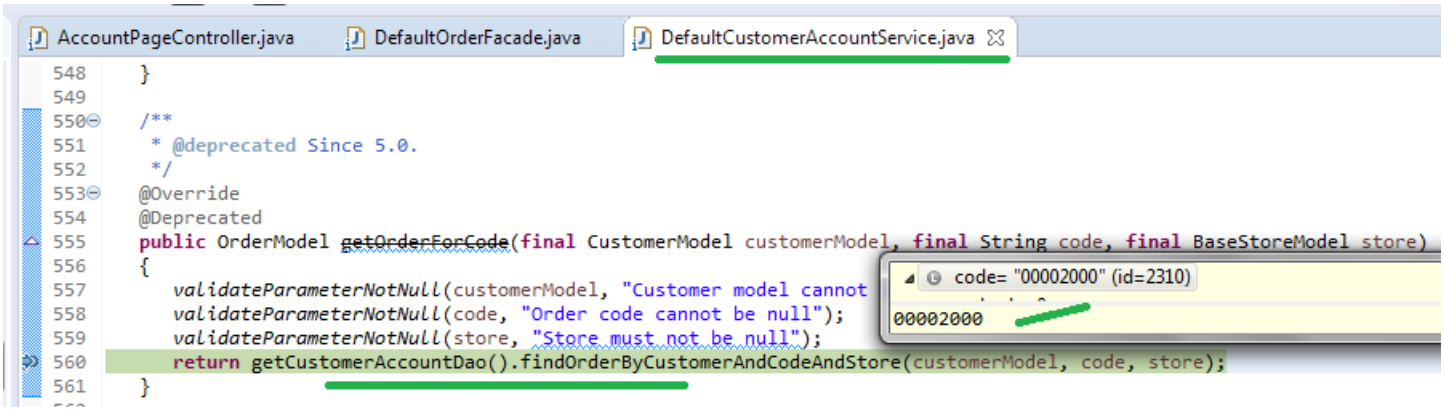
For Cart – We have cartFacade

For Order – We have orderFacade

For User – We have userFacade



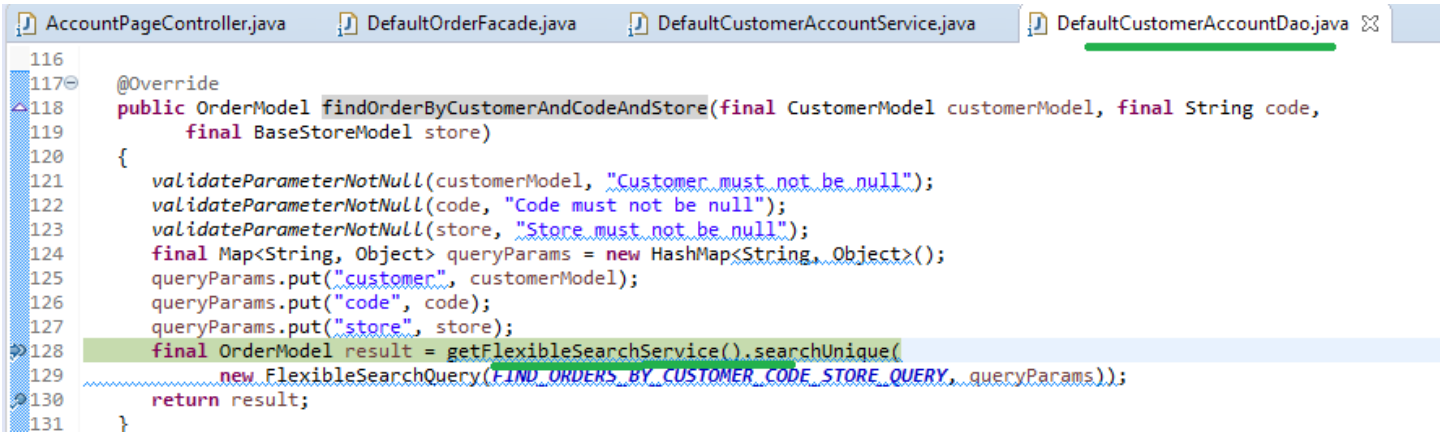
Step 4 = Service Layer



Step 5 = DAO Layer

In DAO Layer – We write Flexible Search Queries

Oracle --- Select == “SAP Comm” – FSQ



Q: How to execute the Flexible Search Queries? = FlexibleSearchService

Note =

For **DB interaction purpose** – We have Model Obj.

For **Storefront interaction purpose** – We have Data Obj.

General Syntax is ➔

If itemtype = **XXX** then we have **XXX**Model & **XXX**Data

For Cart – We have CartModel & CartData

For Order – We have OrderModel & OrderData =====

Note = Above Results (OrderModel) – Return to Service Layer … Service Layer returns to Façade Layer…

Step 6 = Once you get the results into Façade Layer…

Now it's time to convert **OrderModel** into **OrderData**.

That means, General Syntax = **XXX**Model into **XXX**Data.

Q = How to convert Model Obj into Data Obj? = We can use **Converters**.

getOrderConverter().**convert**(orderModel);

General Syntax = **get**XXX**Converter().convert(**xxxModel);

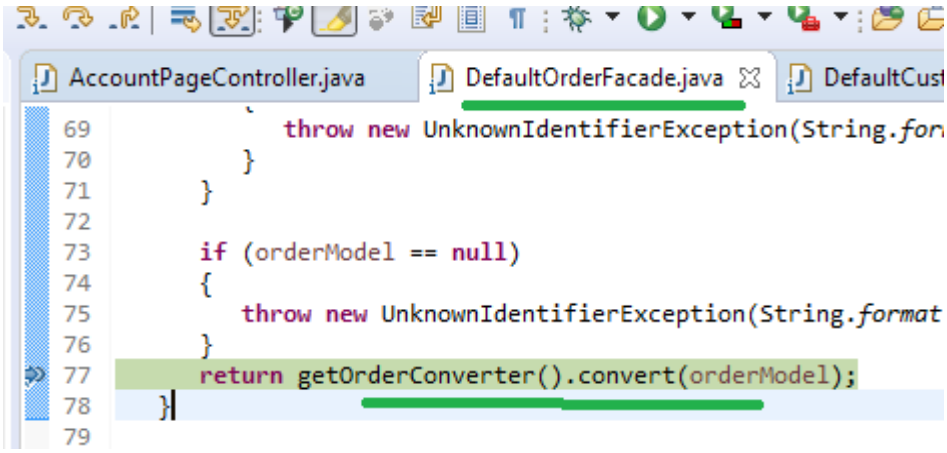
Note = Converter internally calls the **populator**.

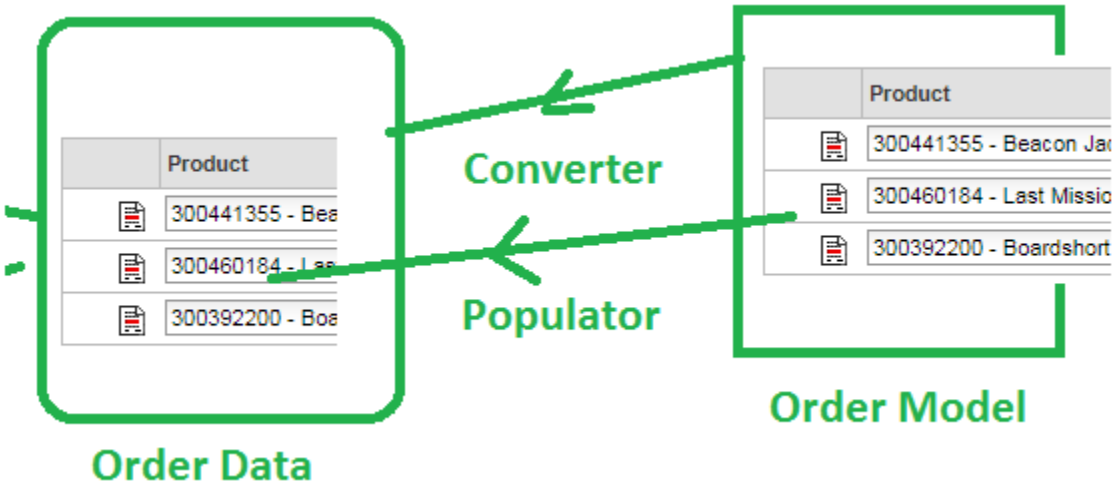
Q = What is the purpose of the Populator?

Populator **populate** the data from **Model obj** into **Data Obj**.

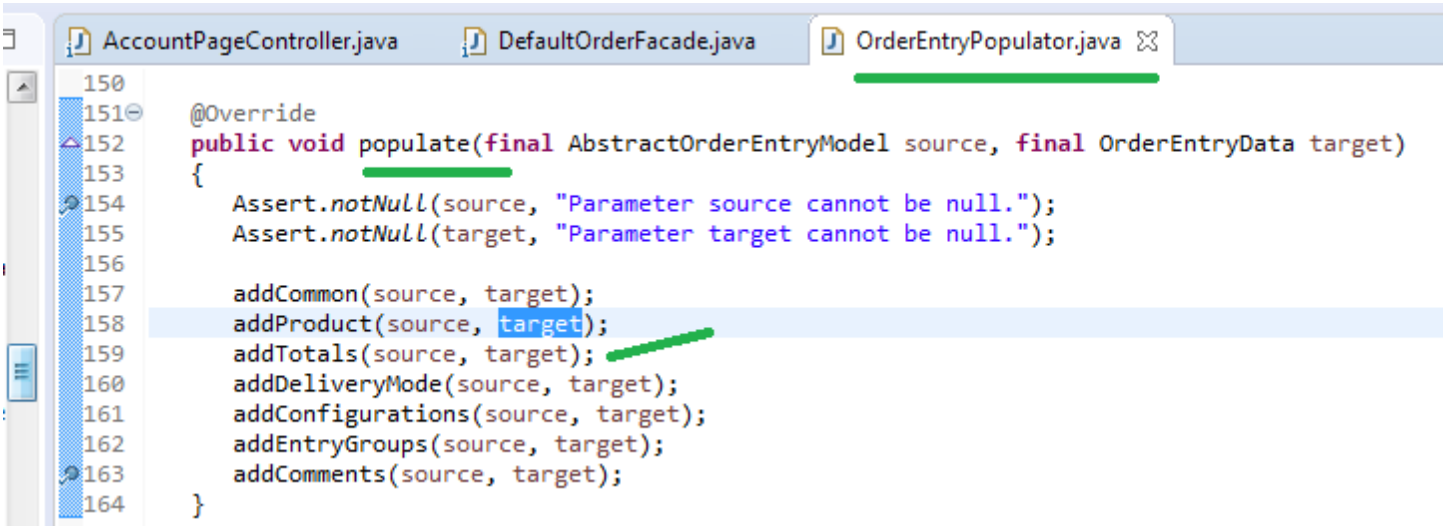
That means, Populator takes data from Model Obj & Put in Data Obj.

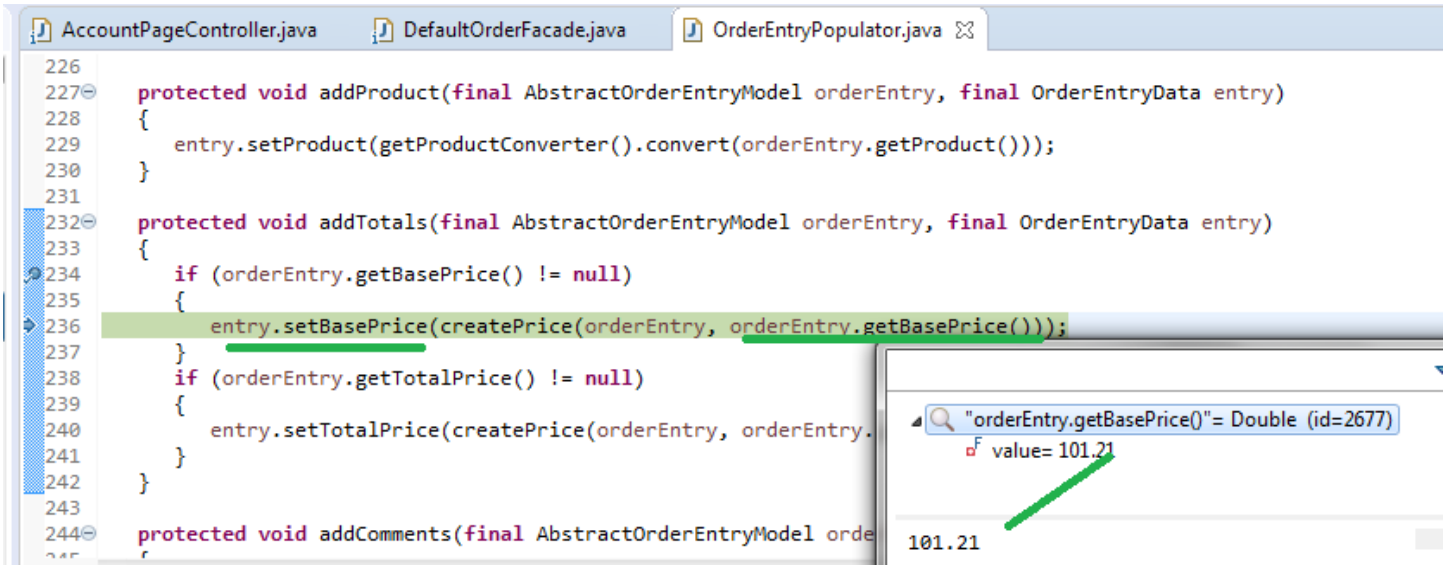
Populator – Will have **Populate()** method.





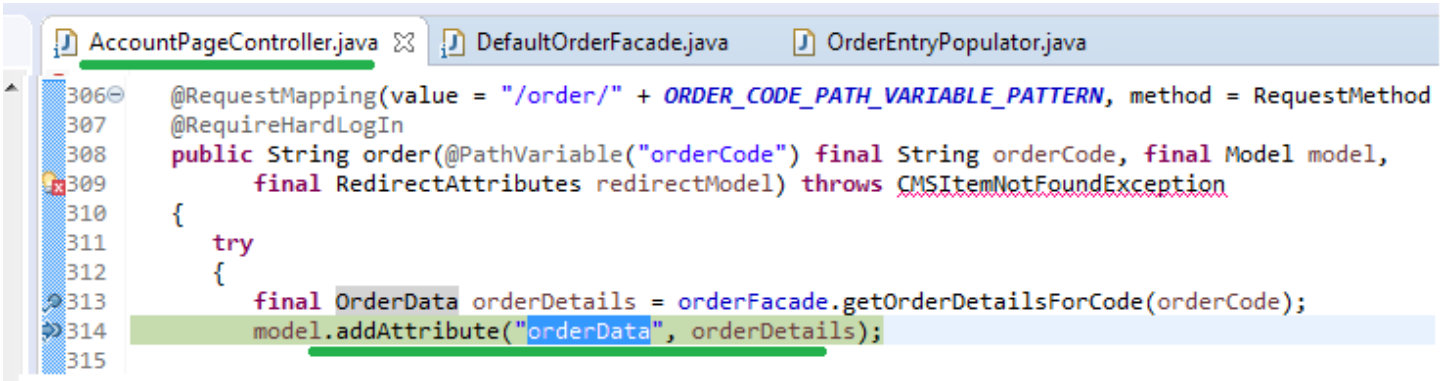
Contact Us = ChennaReddyTraining@RRRS.CO.IN



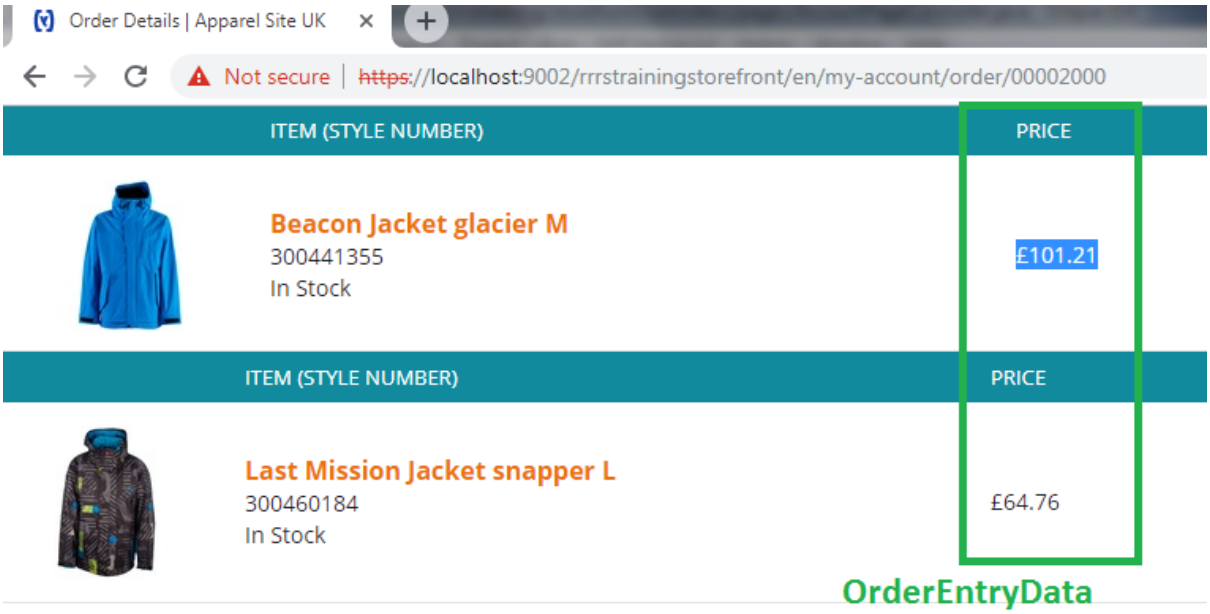


So – Data Obj = **OrderEntryData**

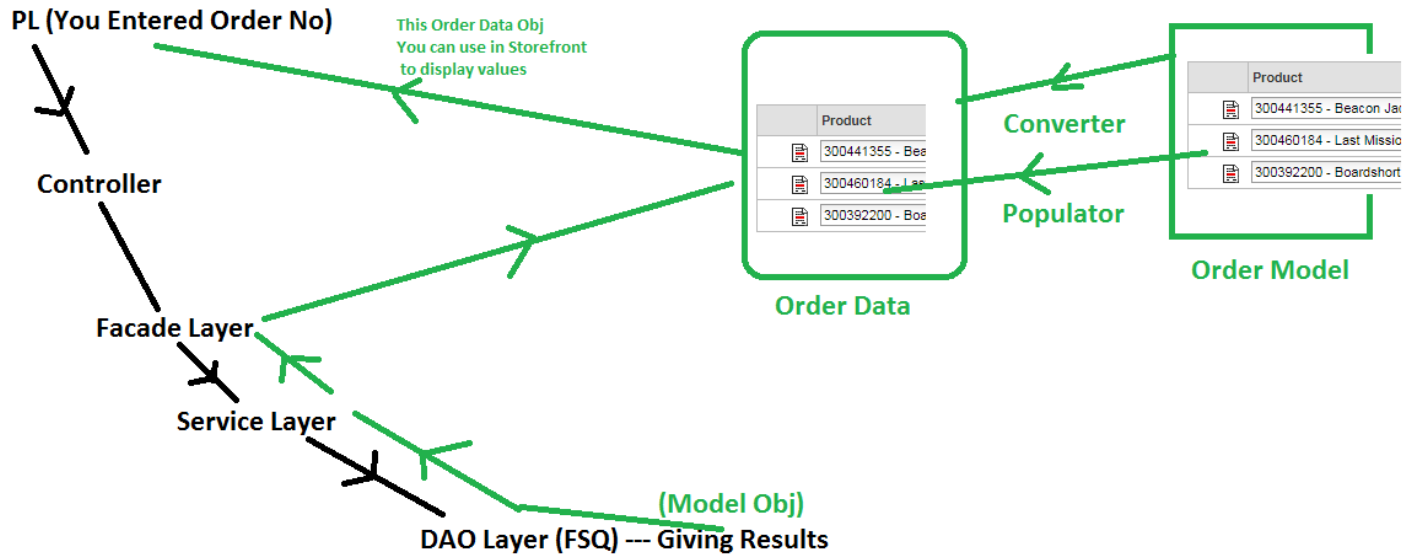
Step 7 = Once – Convertor & Populator are executed, your will flow return to Controller.



Step 8 = Now Use the “OrderData and OrderEntryData” Objects data UI in Layer & display the results.



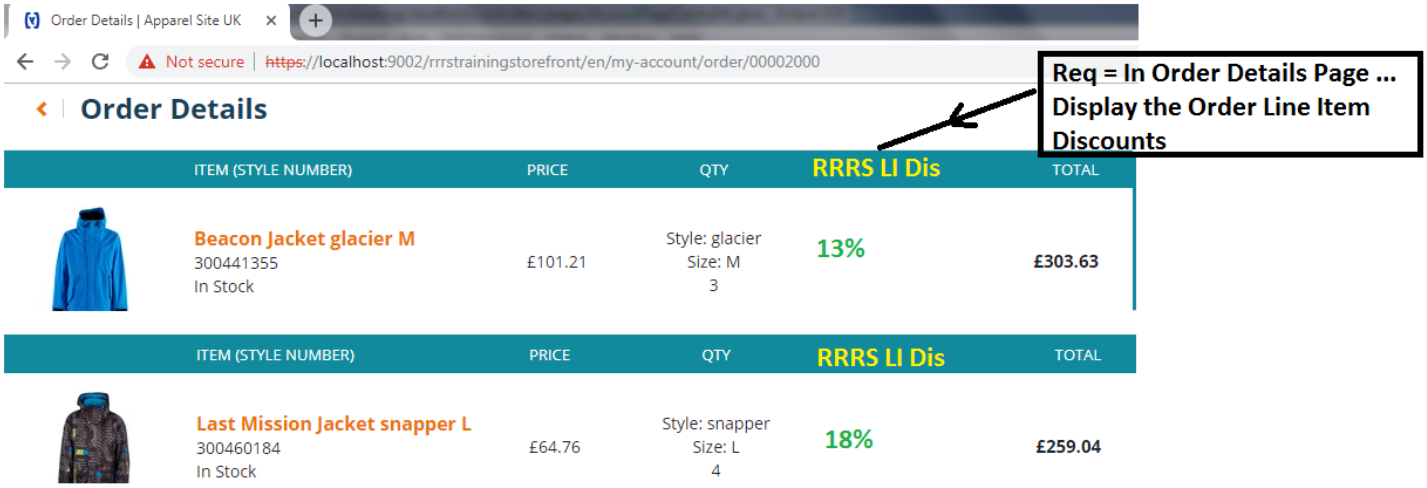
Note: - Let’s verify how Data Object (**OrderEntryData**) is used in UI Layer?

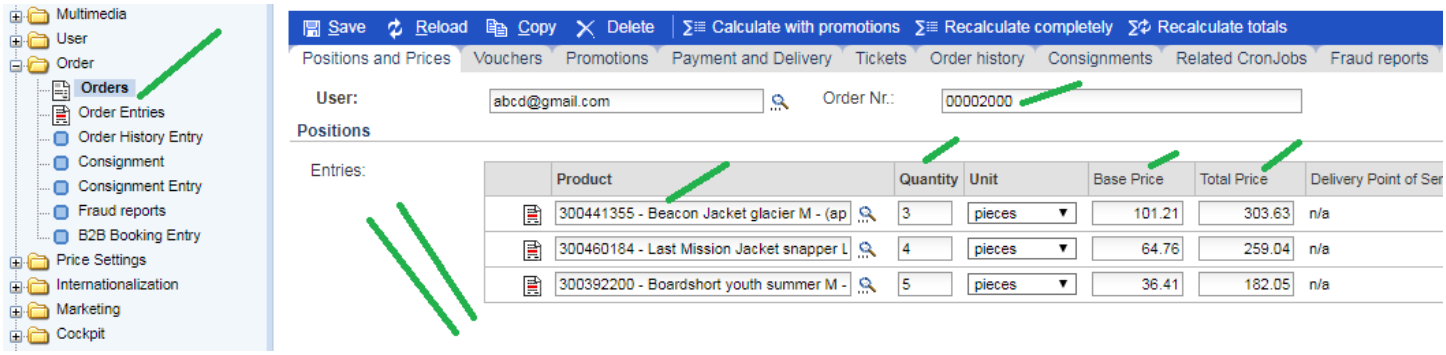


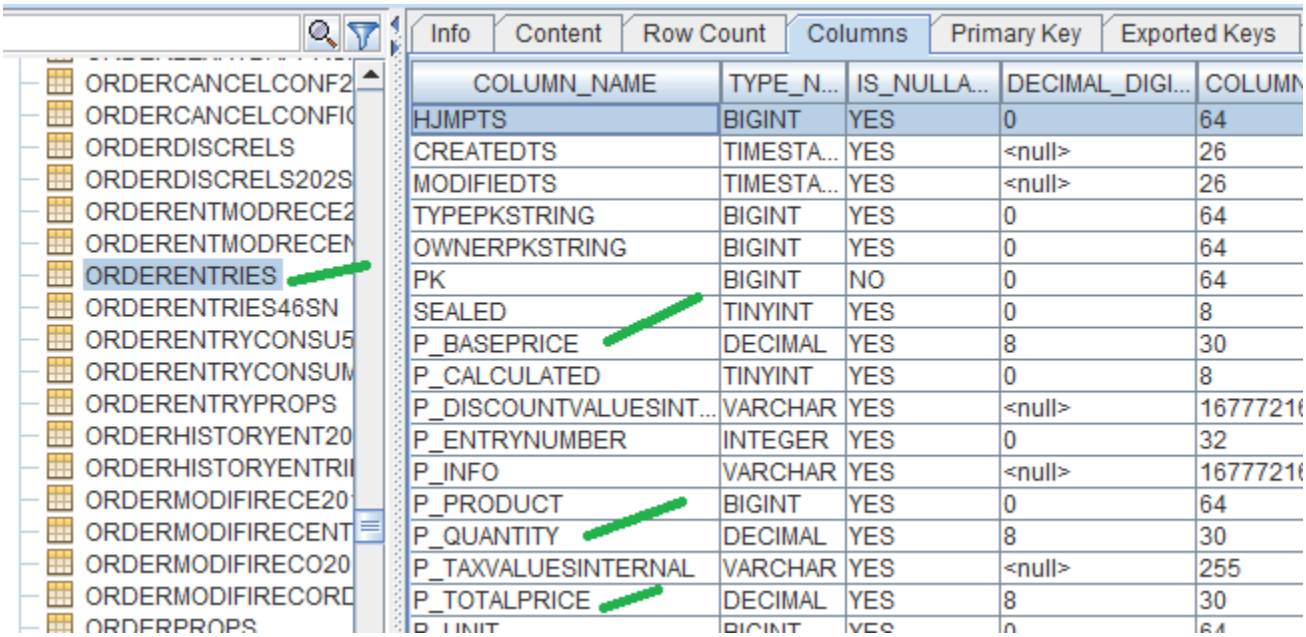
Contact Us = ChennaReddyTraining@RRRS.CO.IN

Business Request = In Order Details Page, display "Order Line Item Discount (Example: - 13% … 18% …)" from DB field.

Also, display Order Line Item Label = **RRRS LI Dis.**







Contact Us = ChennaReddyTraining@RRRS.CO.IN

Step 1 = Create New Colum called "**RRRSLIDis**"

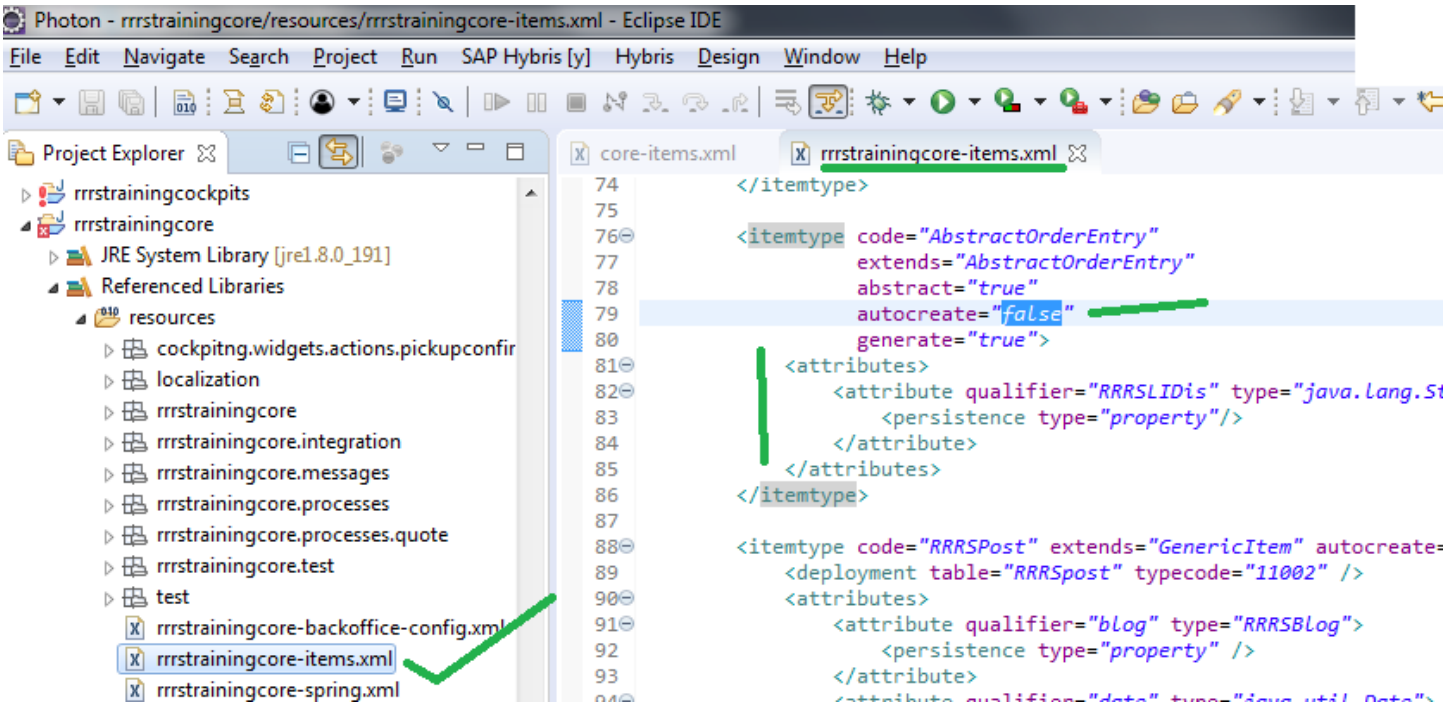
core-items.xml

```
2527        <itemtype code="AbstractOrderEntry"
2528                  extends="GenericItem"
2529                  jaloclass="de.hybris.platform.jalo.order.AbstractOrderEntry"
2530                  abstract="true"
2531                  autocreate="true"
2532                  generate="true">
2533            <attributes>
2534                <attribute autocreate="true" qualifier="basePrice" type="java.lang.Double"
2535                    <persistence type="property">
2536                        <columntype>
2537                            <value>java.math.BigDecimal</value>
2538                        </columntype>
2539                    </persistence>
2540                    <modifiers read="true" write="true" search="false" optional="true"/>
2541                    <defaultvalue>Double.valueOf(0.0d)</defaultvalue>
2542                </attribute>
```

Q = Can we go & add new attribute directly in this "**core-items.xml**"? = Yes, we can do. But this is **not best practice**.

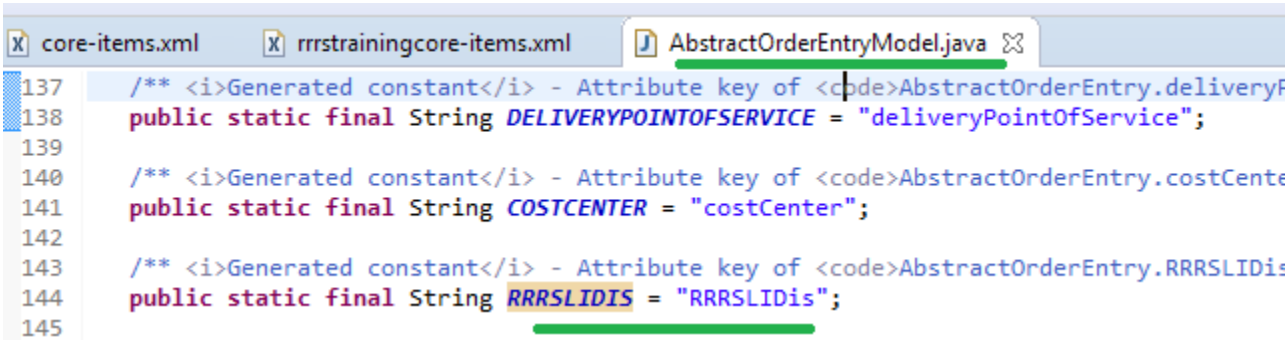
Why this is not best practice? = "**core-items.xml**" file is "SAP Comm" provided & available in platform folder. If we do the changes here, it will work, but tomorrow if any upgrade happens then our **changes will be lost.**

Q = Where can we do the changes (creating new DB field)?



Step 2 = Do the build (**ant clean all**).

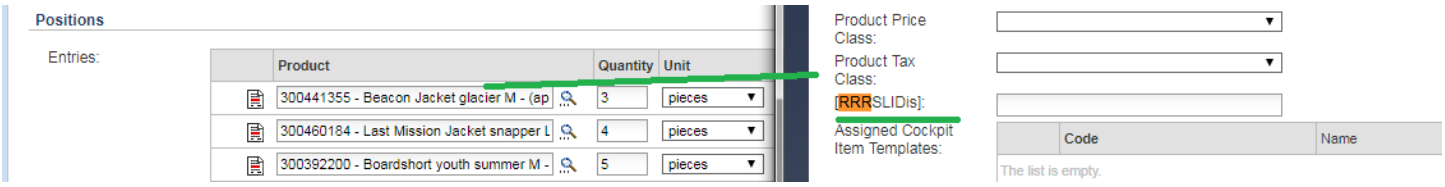
After completion of build – If you open the Model class, it should your attribute.



Step 3 = Start the Server (**hybrisserver.bat**)

Step 4 = Perform the Platform – **hAC Update**

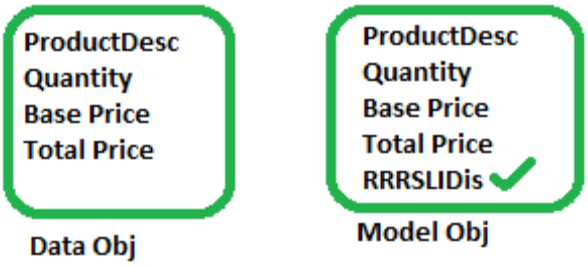
Step 5 = After the update successful – Check the results (New field created in DB).



Note = RRRSLIDis values we need to display in Storefront.

For **DB** interaction purpose – We have **Model Obj** (or) Model Class.

For **PL** interaction purpose – We have **Data Obj** (or) Data Class.

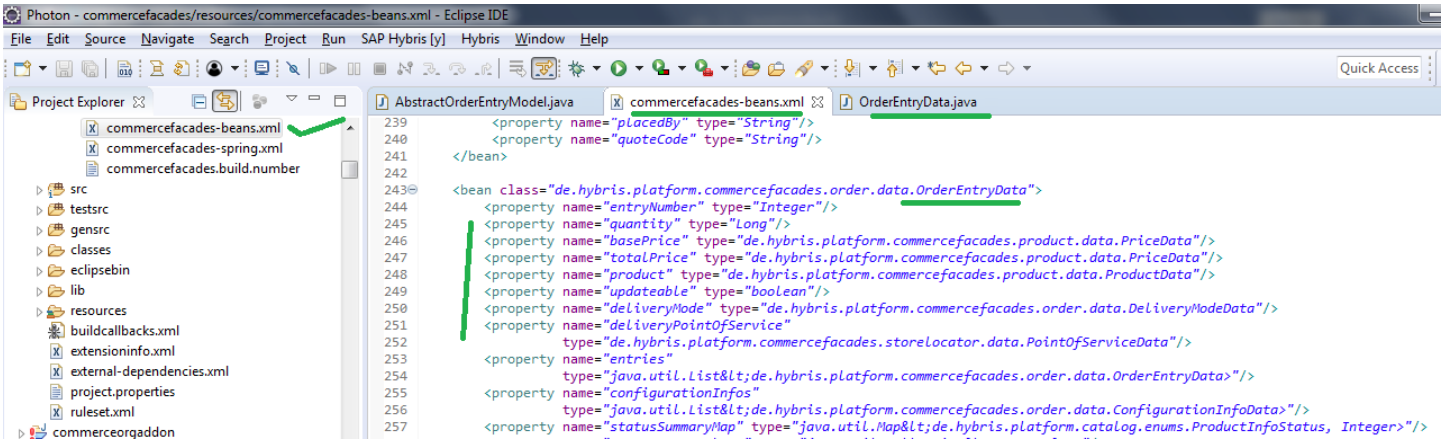


Note = So, now it's time to create attribute in "**Data class / Data Obj**".

If we want new attribute in **Model class** **(or) Model Obj** – then we need to do the changes in "***core-items.xml**".

If we want new attribute in **Data Class** **(or) Data Obj** – then we need to do the changes in "***facades-beans.xml**".

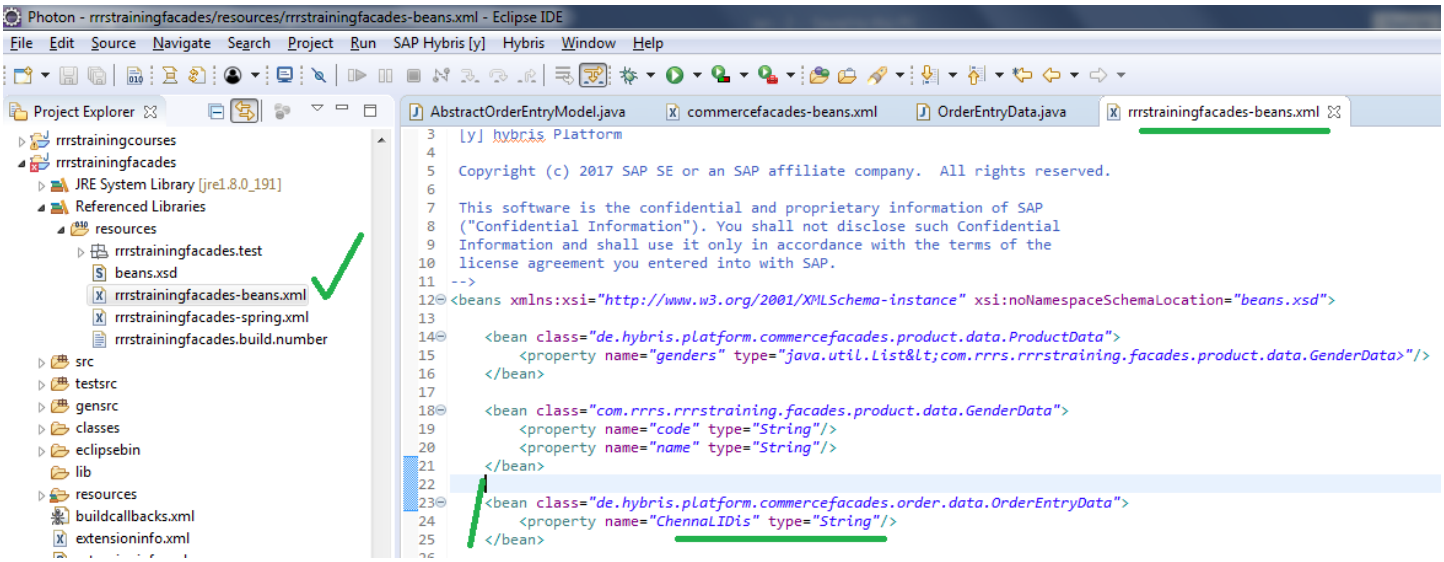
Step 6 = Create new attribute called "**ChennaLIDis**" in Data class (or) Data Obj.



Q = Can we go & add new attribute directly in this "**commercefacades-beans.xml**"? = Yes, we can do. But this is **not best practice**.

Why this is not best practice? = "**commercefacades-beans.xml**" file is "SAP Comm" provided. If we do the changes here, it will work, but tomorrow if any upgrade happens then our changes will be lost.

Q = Where can we do the changes?

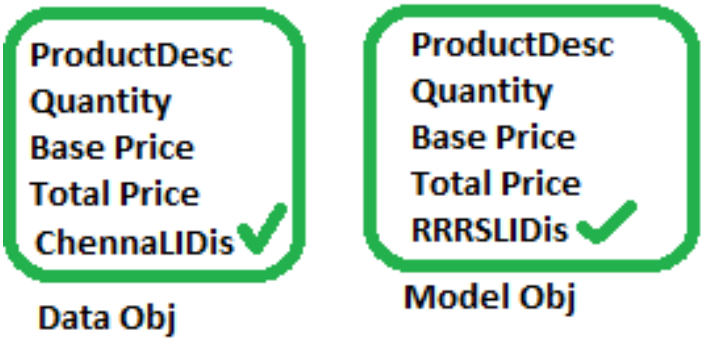


Note = Model Obj (or) Model class & Data Obj (or) Data class can have same attribute names.

Step 7 = Do the build (**ant clean all**)

Note = After build successful –

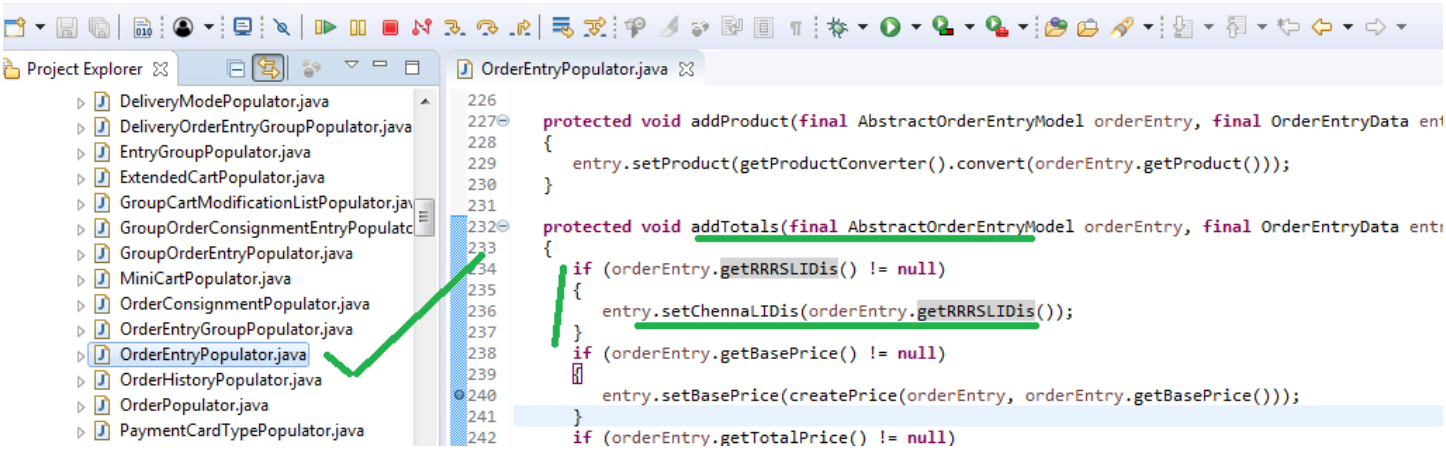
Check results (New Attribute ChennaLIDis created).



OrderEntryPopulator.java | OrderEntryData.java

```
108    /** <i>Generated property</i> for <code>OrderEntryData.cancelledItemsPrice</code> pro
109
110    private PriceData cancelledItemsPrice;
111
112    /** <i>Generated property</i> for <code>OrderEntryData.returnedItemsPrice</code> prop
113
114    private PriceData returnedItemsPrice;
115
116    /** <i>Generated property</i> for <code>OrderEntryData.ChennaLIDis</code> property de
117
118    private String ChennaLIDis;
119
```

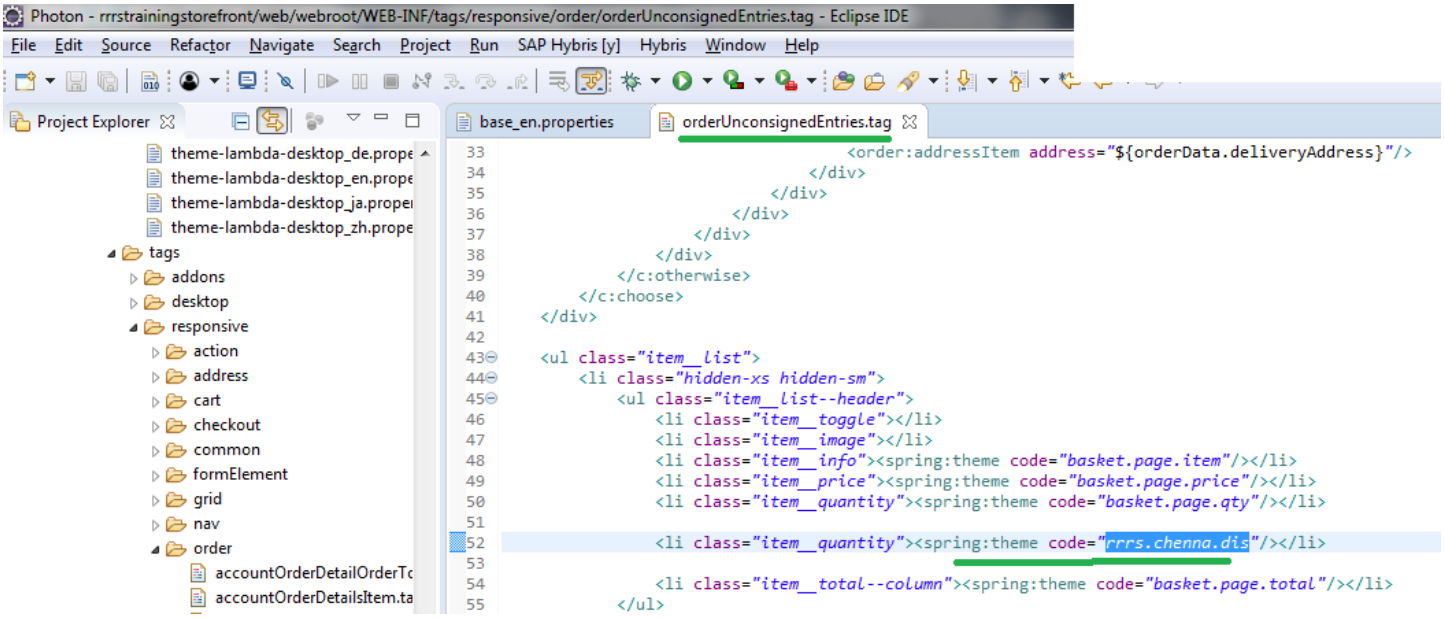
Step 8 = Populate the Model Obj data into Data Obj.

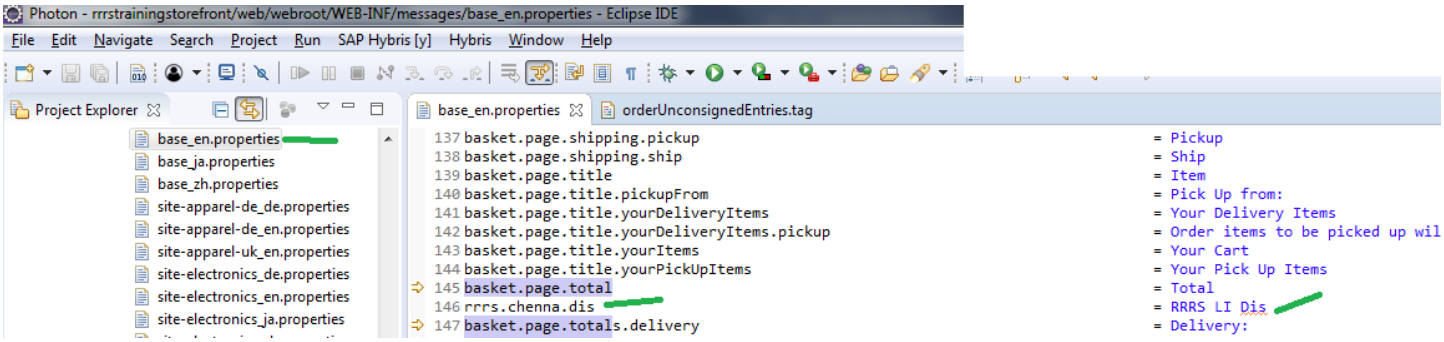


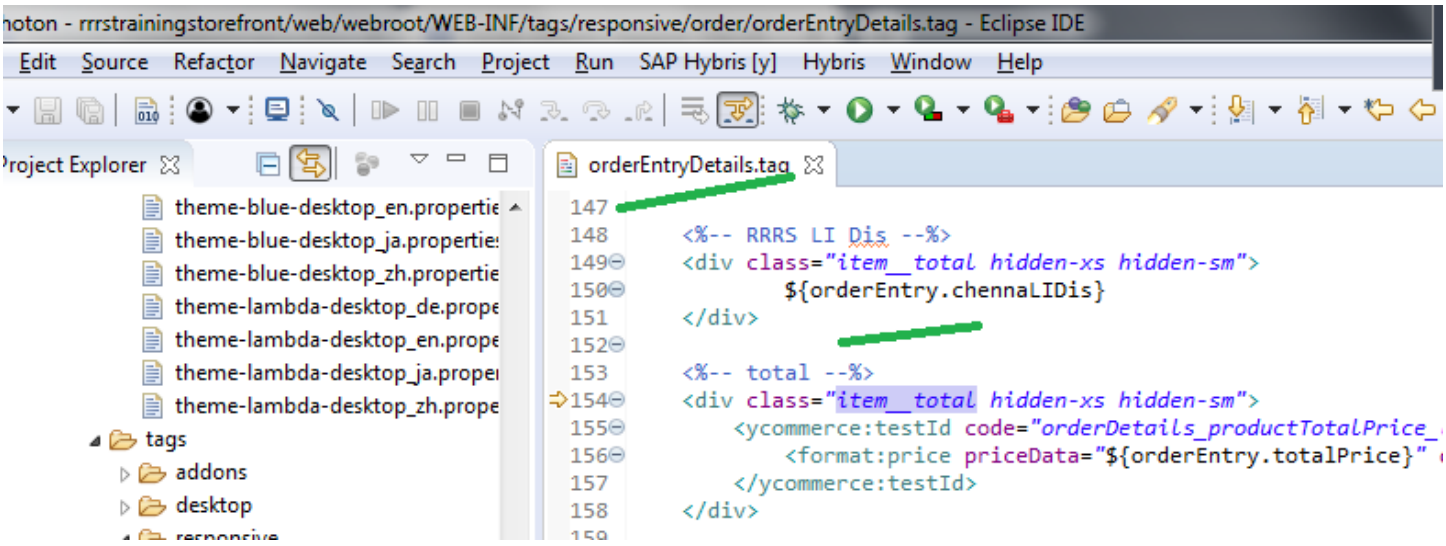
Step 9 = Do the build (**ant clean all**)

Step 10 = Start the Server (**hybrisserver.bat**)

Step 11 = Do the UI (or) PL layer changes.







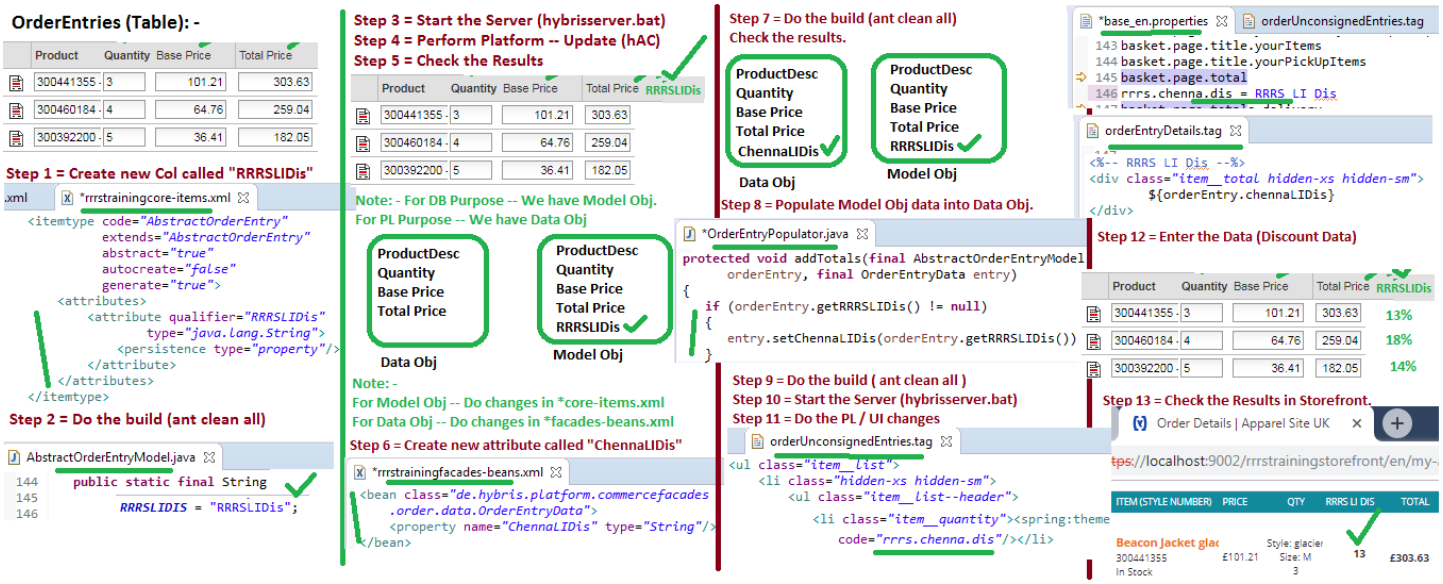
Contact Us = ChennaReddyTraining@RRRS.CO.IN

Step 12 = Enter Discount Data

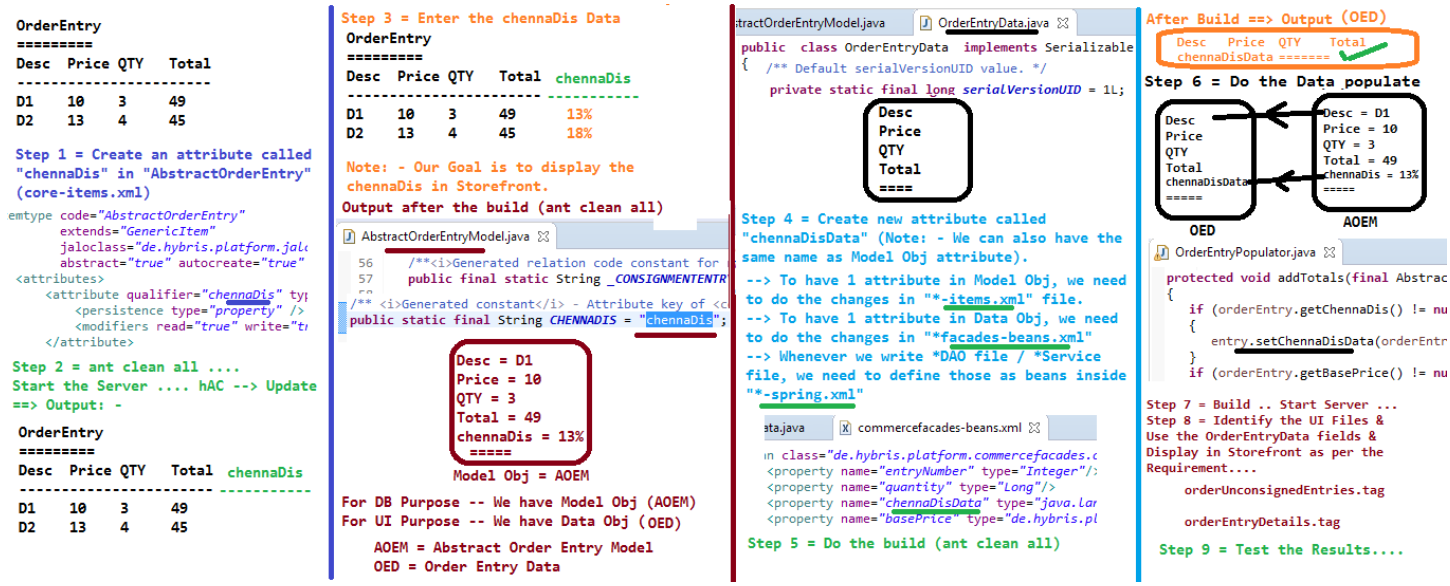
	Product	Quantity	Base Price	Total Price	RRRSLIDis
	300441355 -	3	101.21	303.63	13%
	300460184 -	4	64.76	259.04	18%
	300392200 -	5	36.41	182.05	14%

Step 13 = Test the Results in Storefront now: -

See Overall Steps: -



Contact Us = ChennaReddyTraining@RRRS.CO.IN



Conclusion = In this scenario –

We already have the table and columns.

We already have the “SAP Comm” code [DAO File … Service File … Façade File … Converter … Populator … Controller …]

We already have the “UI Code”.

Contact Us = ChennaReddyTraining@RRRS.CO.IN

Then what we are trying to do?

We created new column & display that column data in UI layer.

High level steps =

Step 1 = Create new column in existing table [*core-items.xml]

Step 2 = Create new variable in Data Obj [*facades-beans.xml]

Step 3 = Do Populator changes

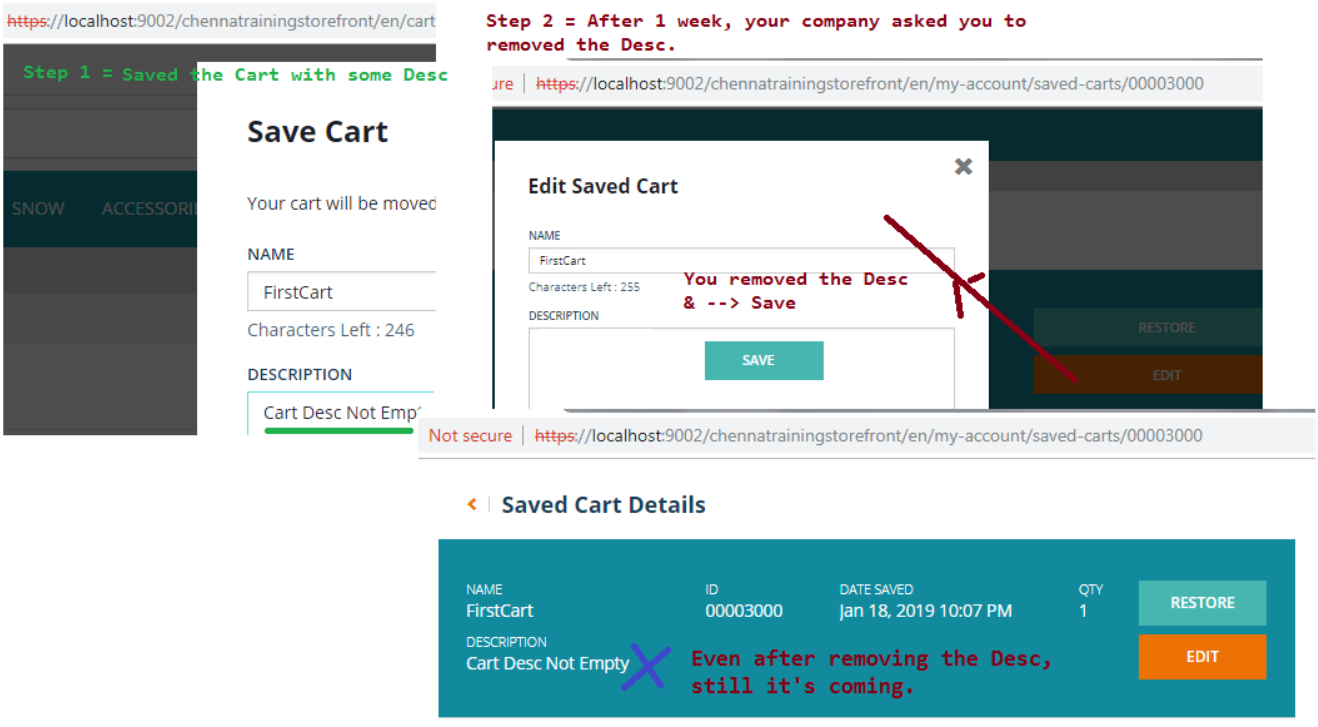
Step 4 = Do UI Changes

Step 5 = Start the Server … ”hAC – Update”.

Step 6 = Results [Test it]

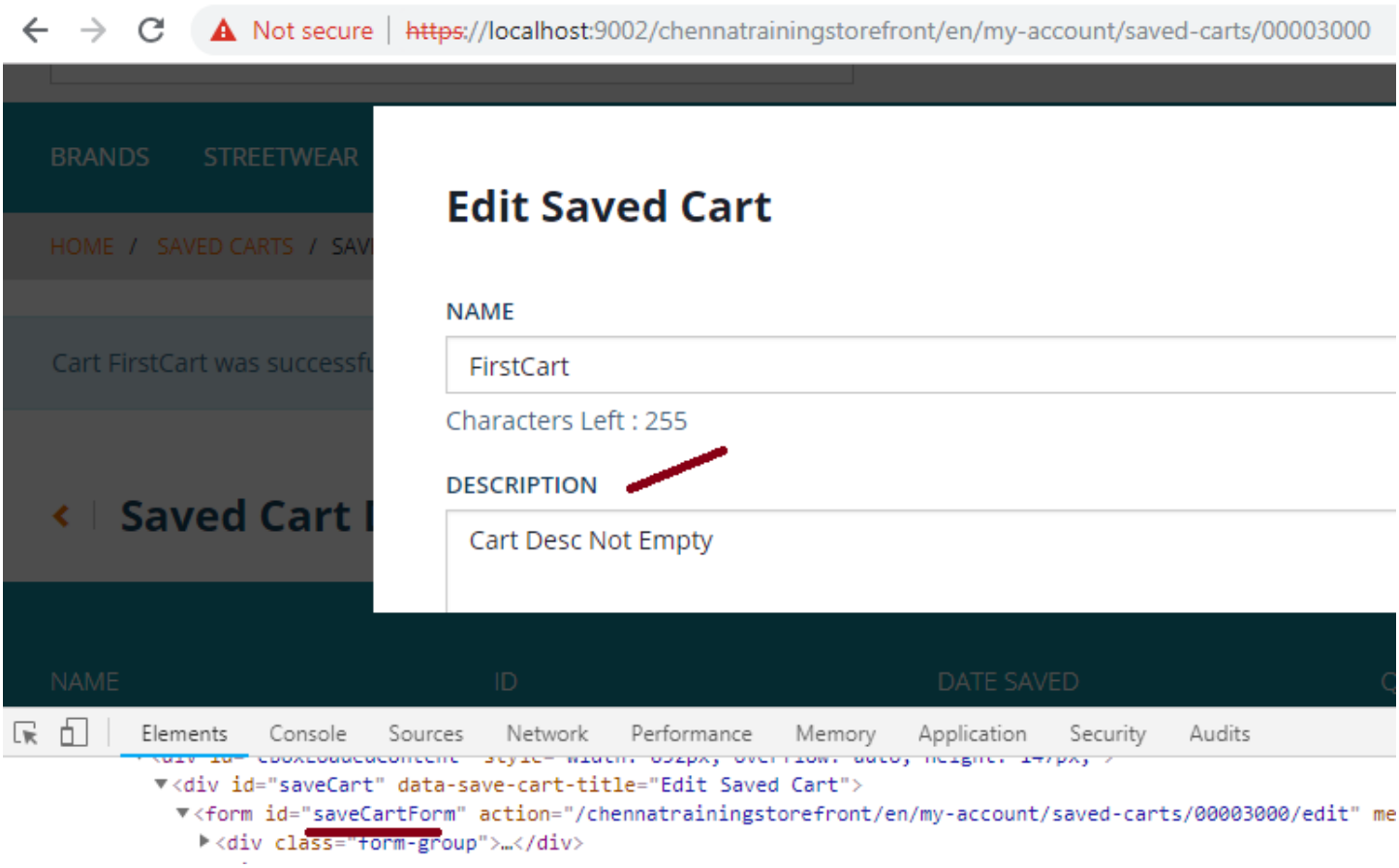
Business Requirement =

Save Cart – Wil have Description. Let’s say we entered some Desc today, after 1 week, if we don’t want then we are not able to remove.

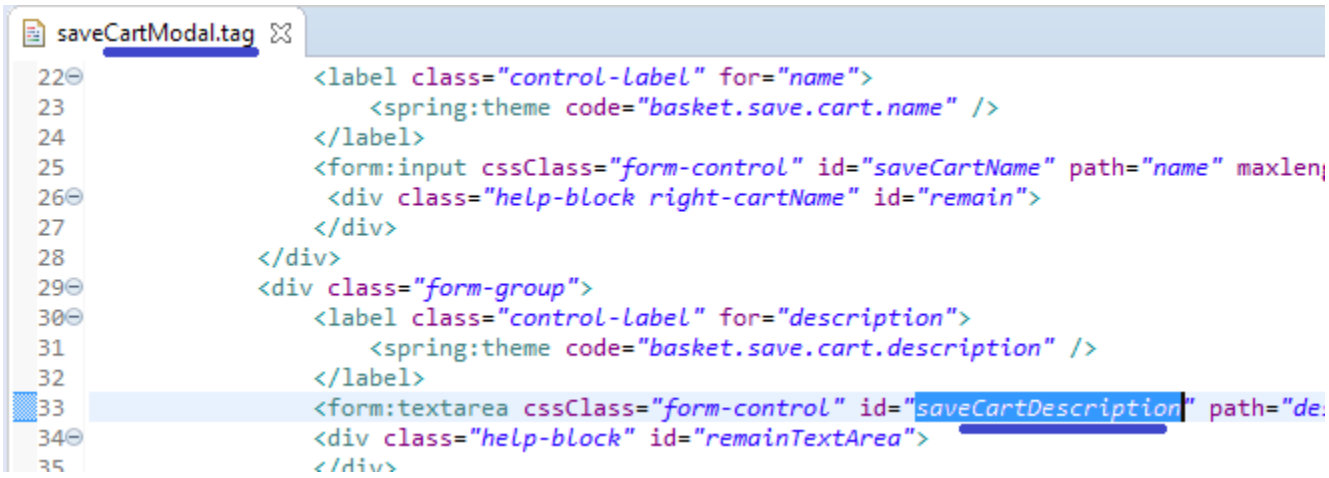


Contact Us = ChennaReddyTraining@RRRS.CO.IN

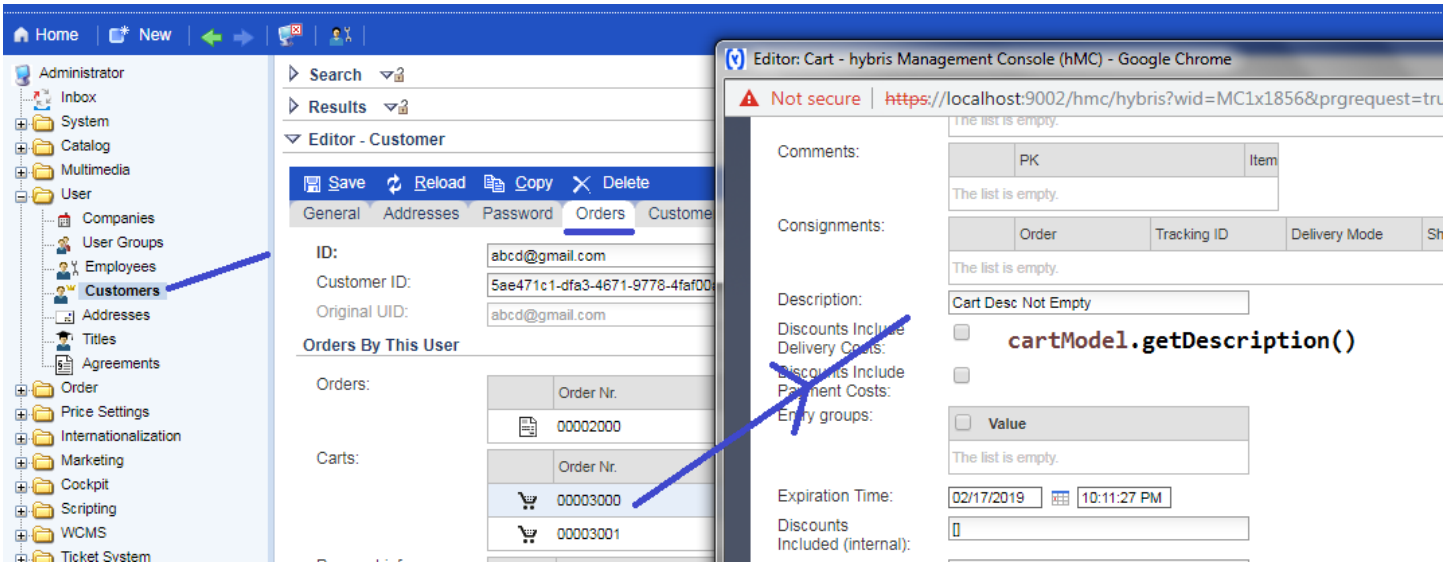
Step 1 = Find the UI code

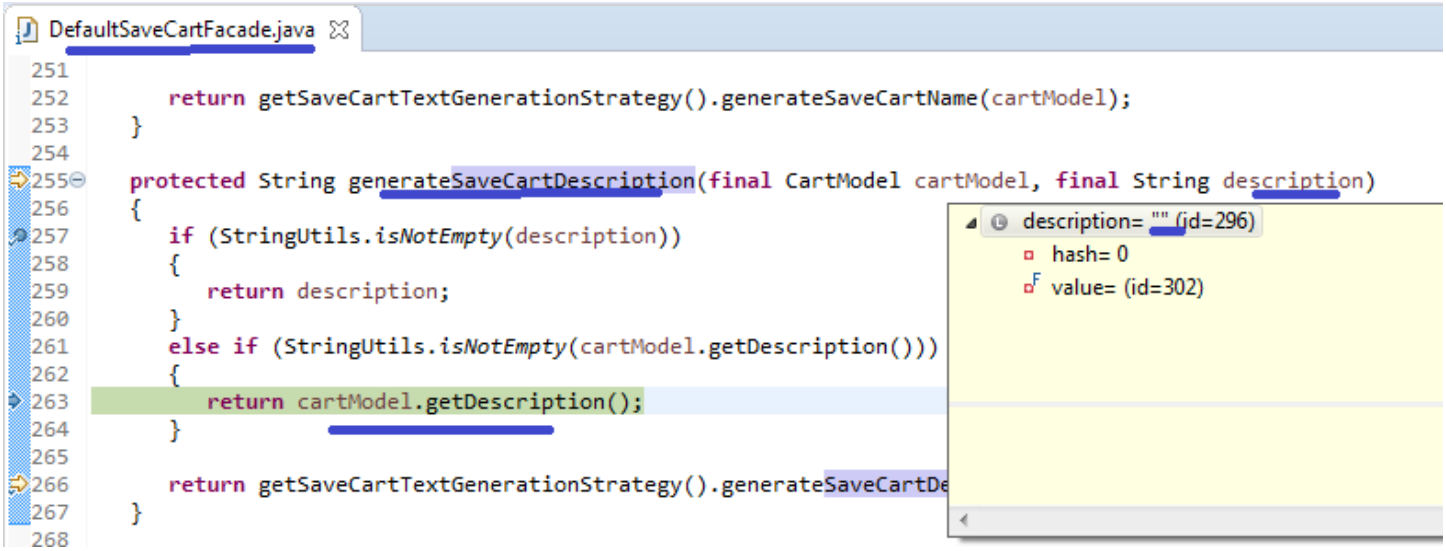


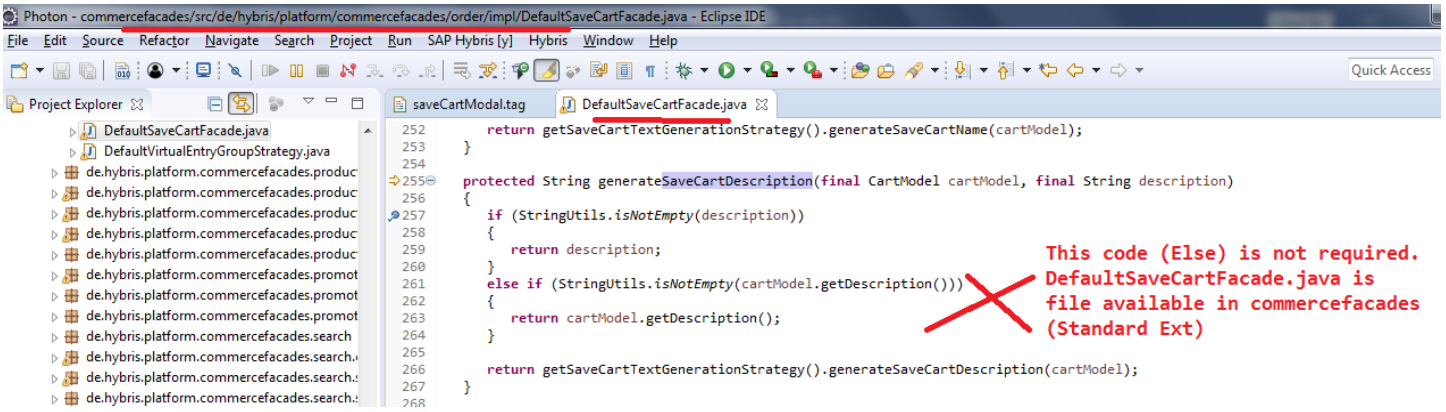
UI File = saveCartModel.tag



Contact Us = ChennaReddyTraining@RRRS.CO.IN







Contact Us = ChennaReddyTraining@RRRS.CO.IN

In Above method Logic Is: -

If Desc (UI Entered value) is not empty, then return the same Desc.

Else we are returning Cart Mode – Desc (Which is previously saved).

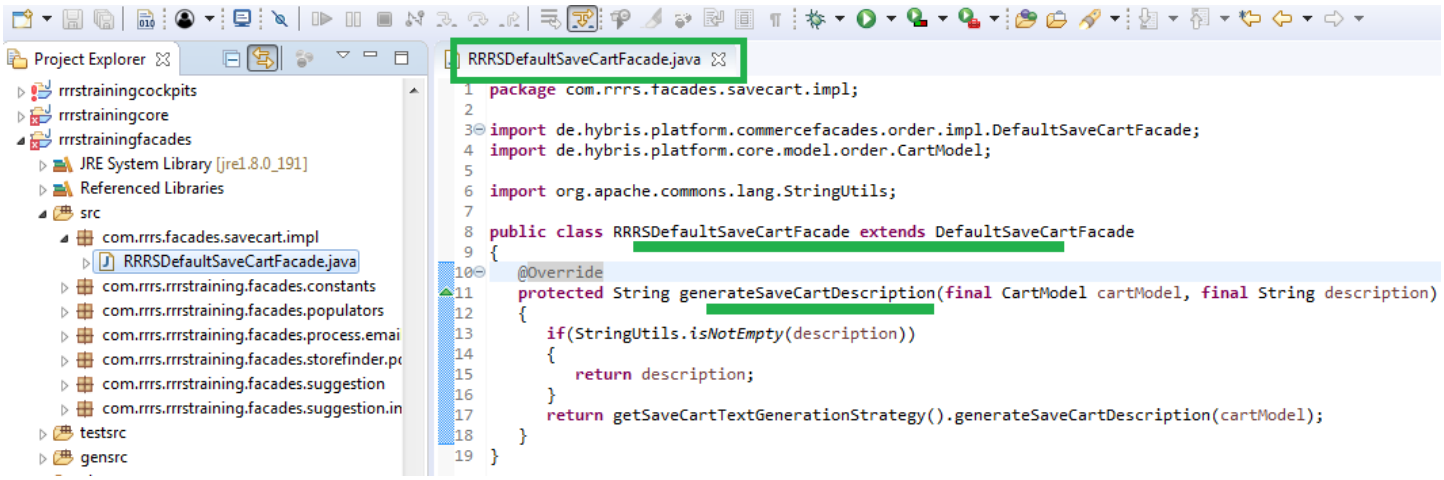
Step 2 = How to fix this.

C:\rrrssoftwares\HYBRISCOMM64\hybris\bin\ext-commerce\commercefacades\src\de\hybris\platform**commercefacades**\order\impl\DefaultSaveCartFacade.java (DSCF)

DSCF.java is available in commercefacades Ext (This is provided by "SAP Comm").

Thumb Rule == Generally we don't modify anything directly inside "SAP Comm" provided platform Exts.

So → U need to Override above Java file methods in your Exts (**rrrstrainingfacade**).

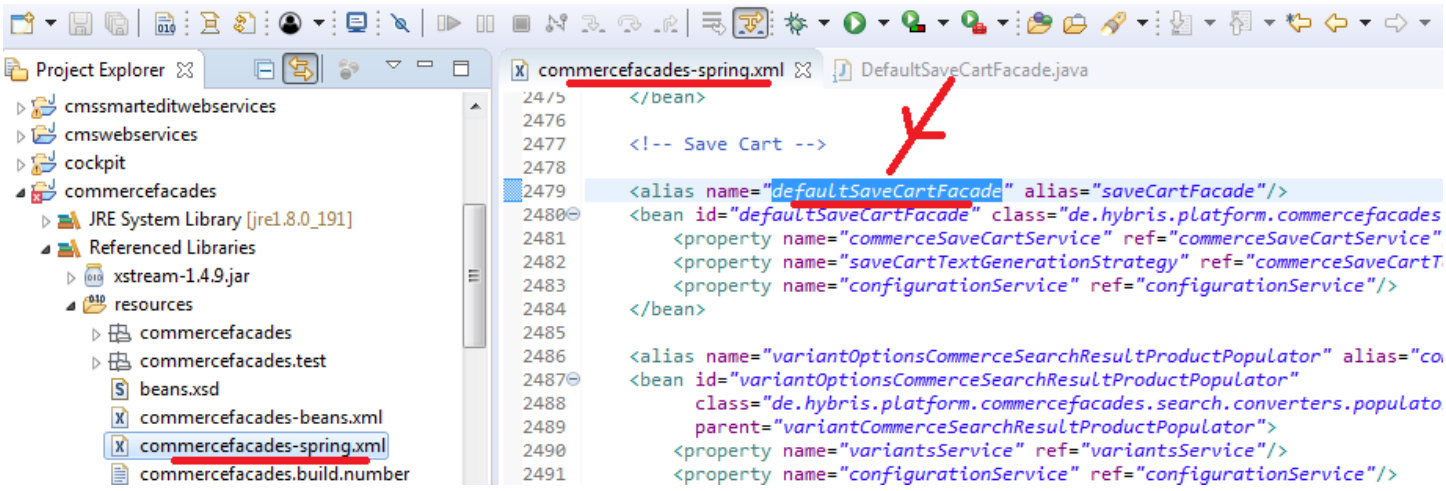


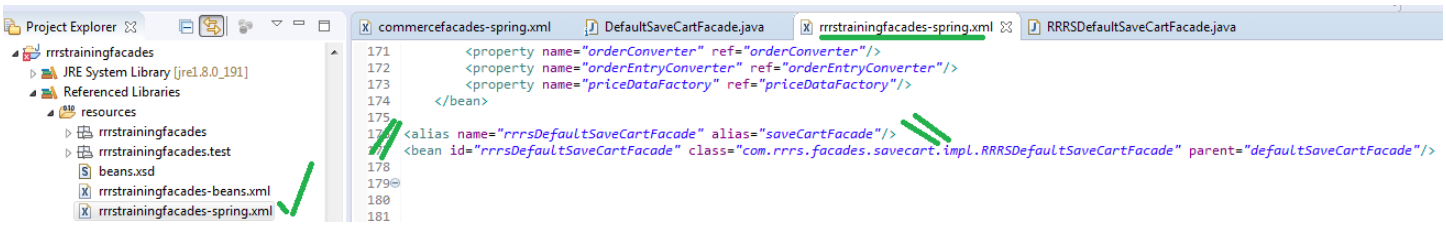
Step 3 = Whenever you write Façade File / Service File / …. [Java File] → You need to define this in ***-spring.xml**

If you want any DB Changes → ***core-items.xml**

If you want any Data Obj Fields → ***facades-beans.xml**

If you write any Service File / Façade File / … →***-spring.xml**

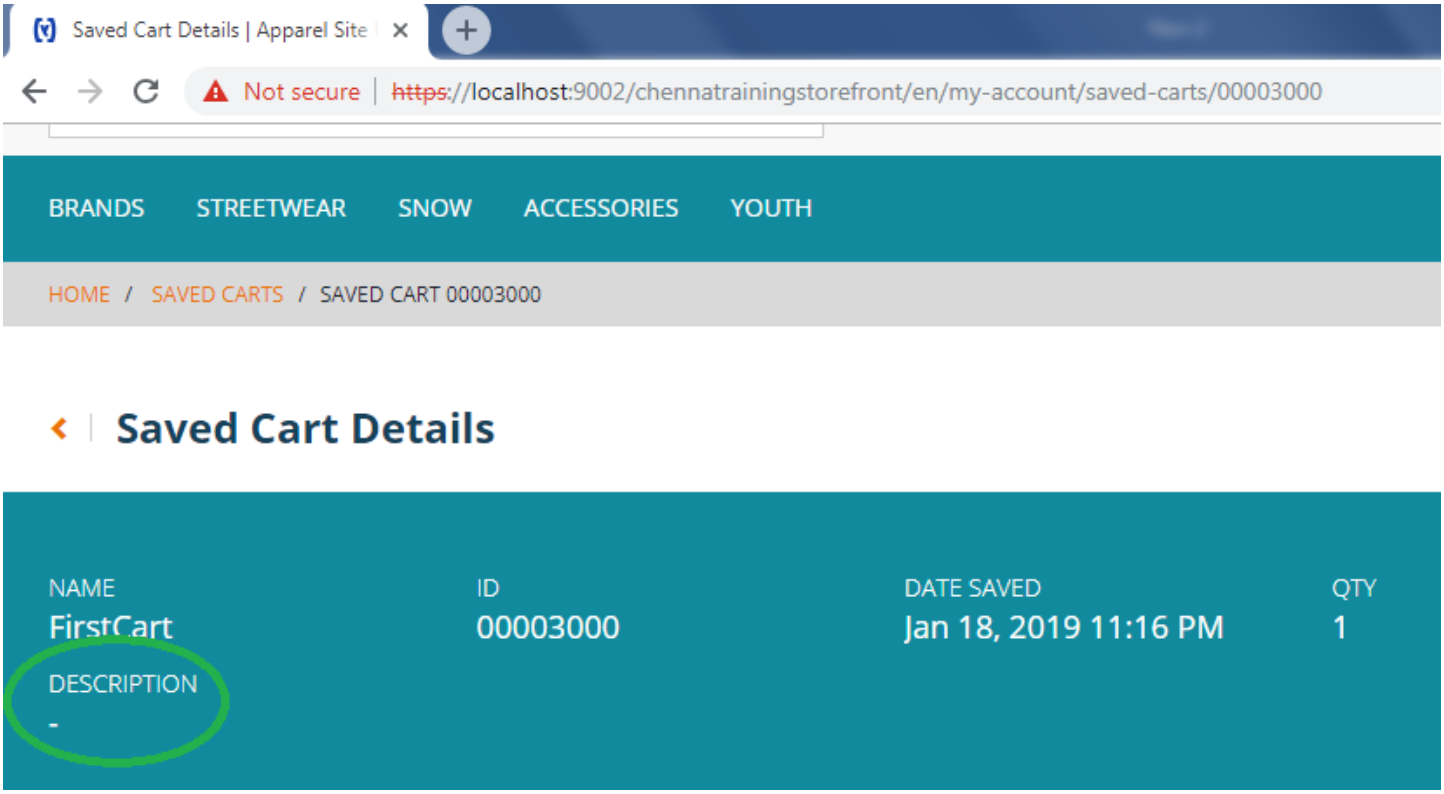




Step 4 = Do the Build (ant clean all)

– Start the Server

– Test the Results.



Conclusion = In this scenario –

We already have the table and columns.

We already have the "SAP Comm" code [DAO File … Service File … Façade File … Converter … Populator … Controller …]

We already have the "UI Code".

Then what we are trying to do?

Some of the logic whatever "SAP Comm" given is not working as per customer needs. Q = How to override the given logic / method?

Step 1 = Find Existing Logic (or) Code

Step 2 = Override the Logic (or) Code (or) Method

Step 3 = Register newly created class [*-spring.xml]

Step 4 = Build … Start … Test

Contact Us = ChennaReddyTraining@RRRS.CO.IN

Q = “SAP Comm” commercefacades extension facades mostly return?

a) Data objects b) Model objects c) Data model d) none

Q = “SAP Comm” ServiceLayer Models should not be used as part of a facade interface, maintaining a clean abstraction of the?

a) Business layer and the persistence layer

b) Business layer and the presentation layer

c) Persistence layer and the presentation layer

d) All

Q: All converters should be Spring configured only and should use the ---------- base class

a) AbstractConverter b) DefaultConverter c) a & b

d) none of above

Q = -------Is an implementation of a Populator pipeline where each population step is evaluated against a Set of Enum values passed by the caller.

a) DefaultConfigurablePopulator b) AbstractPopulatingConverter

c) AbstractConverter

Q = Product Populators are a little unique in that they typically extend a ---------- class that is variant aware and supports the ability of falling back to a variants parent product for attribute values in the event of the source product value being null

a) DefaultProductPopulator b) AbstractProductPopulatingConverter

c) AbstractProductPopulator d) None

Q = ……..extension is the template shipped with the "SAP Comm" Accelerator that you can use as a starting point for your own extension.

a) yacceleratorfacades b) commercefacades

c) both d) none

Q = Data Transfer Objects (DTOs) are objects created to contain….

a) Values b) Business logic c) Both d) none of them

Q = ……..template method that allows a concrete sub-class to pick the appropriate target data object implementation.

a) createListTarget **b) createTarget**

c) createSourceToTarget d)None of the above

Q = Facades are which scoped Spring managed beans

a) yrequest **b) singleton** c) tenant d) request

Q = ……. Provides access to various internationalization switches that the user can make when they visit a specific storefront

a) Store Session façade b) Store Locator façade

c) User façade d) User Locale façade

Q = Explain different types of interceptors?

Model Interceptors = Intercept the behavior of the lifecycle of Models. Model lifecycle consists of loading from Database, saving to Database and deleting or removing from the Database.

In lifecycle of a model, interceptors can intercept & modify model data. It includes auditing, validating & even restricting the data from being removed/deleted from database if certain conditions are not met.

There are 5 types of Interceptors: -

a) LoadInterceptor = Invoked whenever a model is loaded from the database

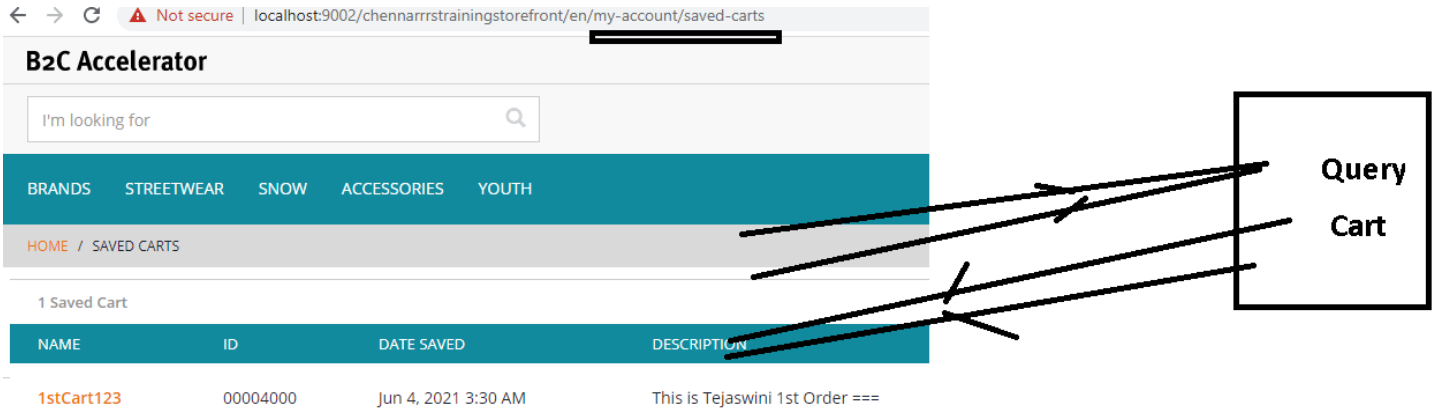
b) InitDefaultsInterceptor = Invoked during modelService.create() & modelService.initDefaults()

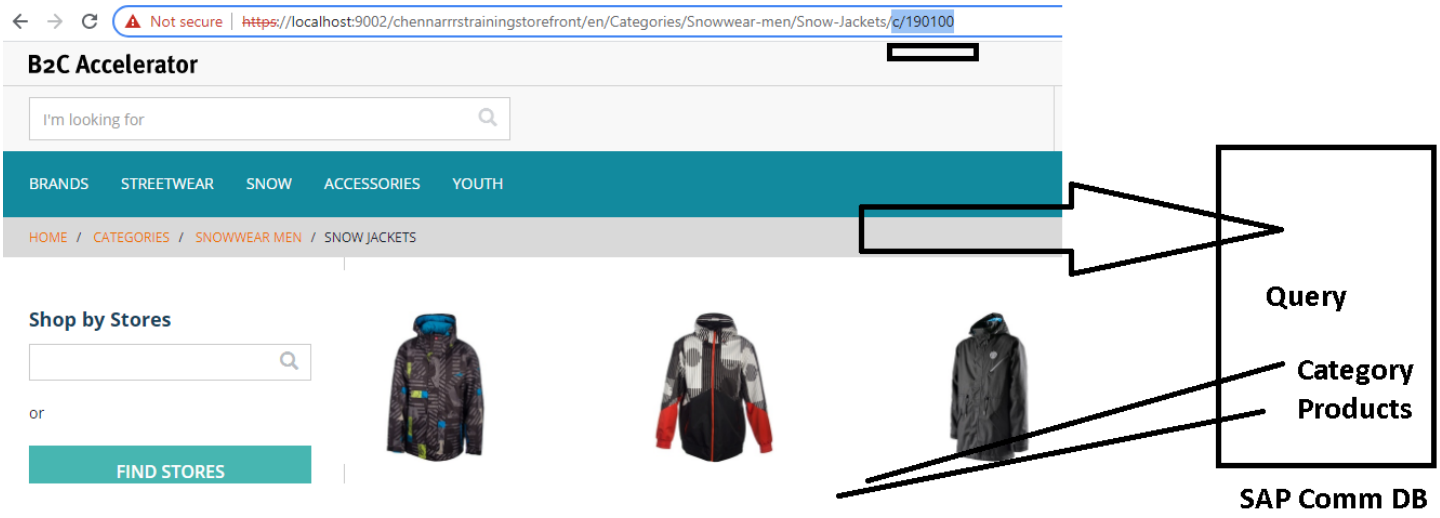
c) PrepareInterceptor = Invoked before model is saved to database & before ValidateInterceptor

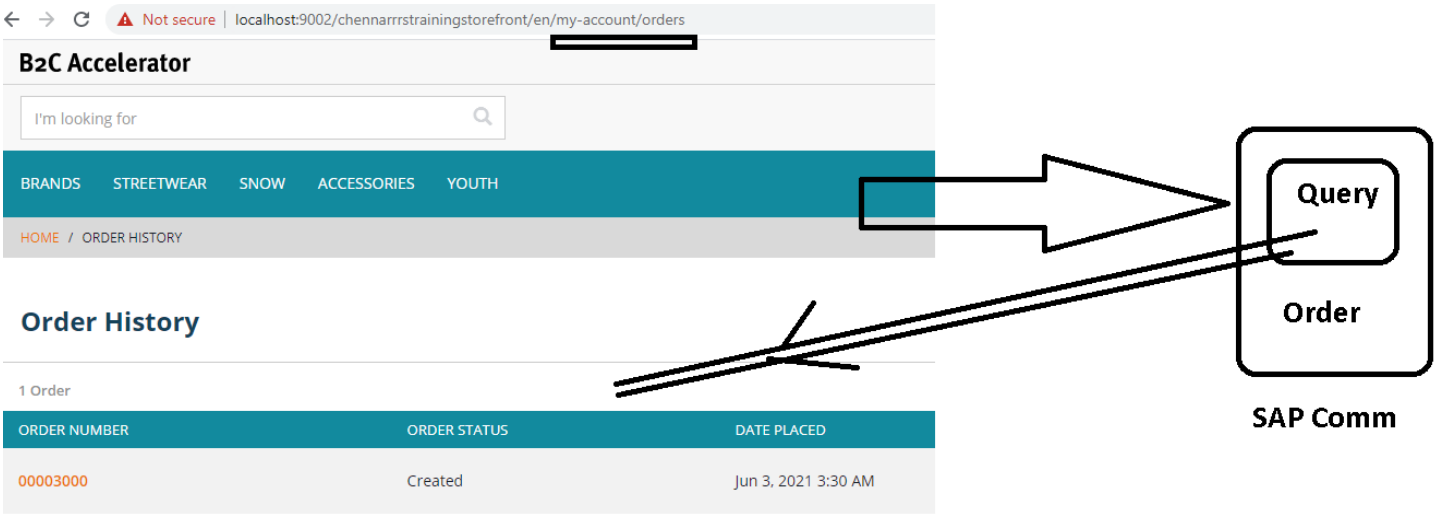
d) ValidateInterceptor = Invoked before a model is saved to database & after PrepareInterceptor

e) RemoveInterceptor = Invoked before a model is removed from database.

End – End Scenarios: -

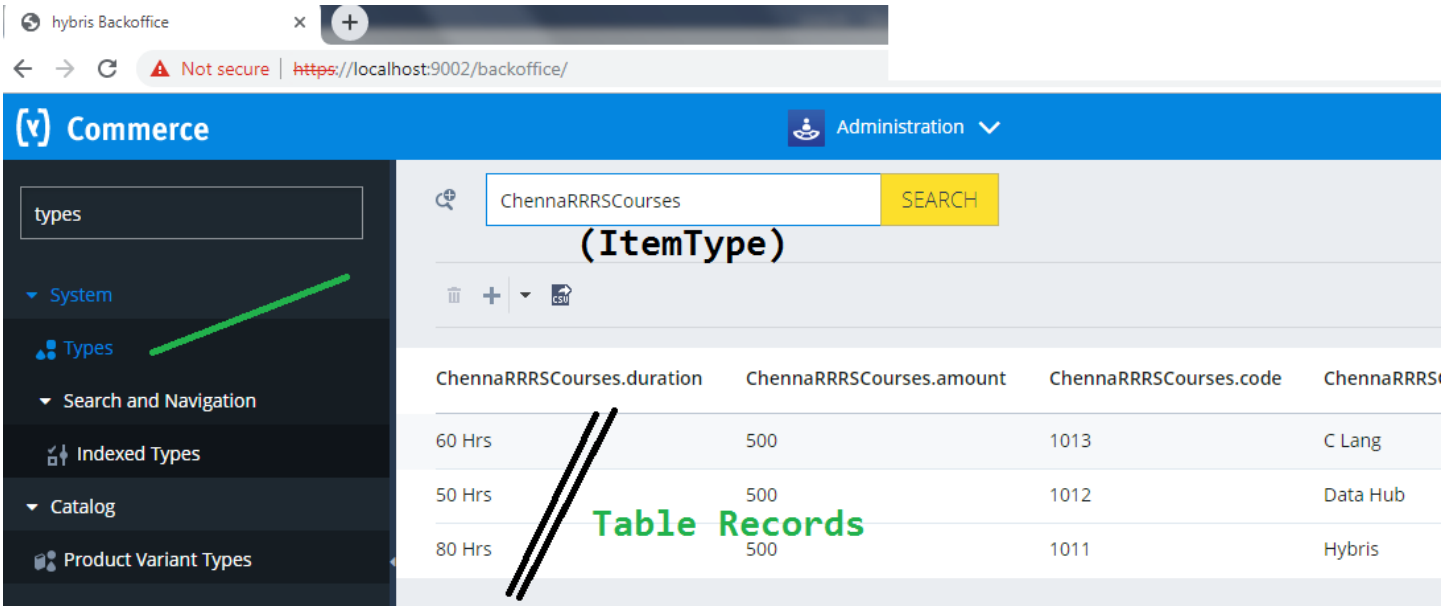




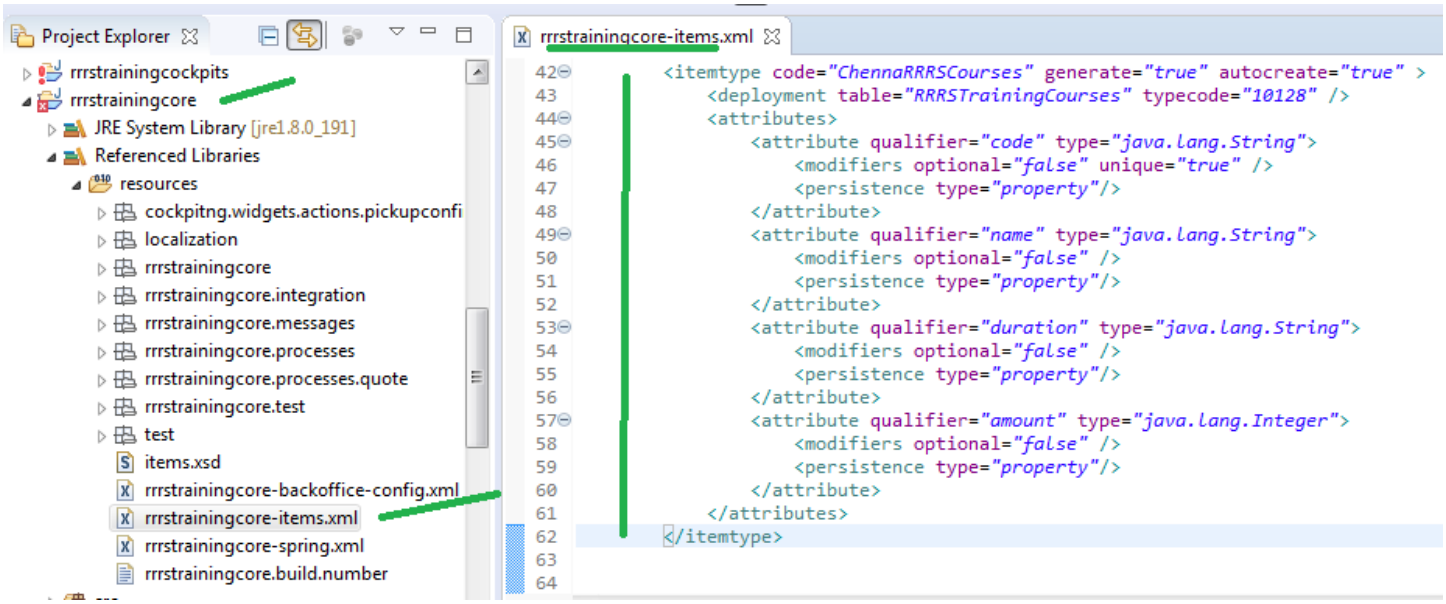


Contact Us = ChennaReddyTraining@RRRS.CO.IN

Requirement = Create an itemtype called "**ChennaRRRSCourses**" & Table name = "**RRRSTrainingCourses".** Insert the Records … Display the records in **Storefront**.

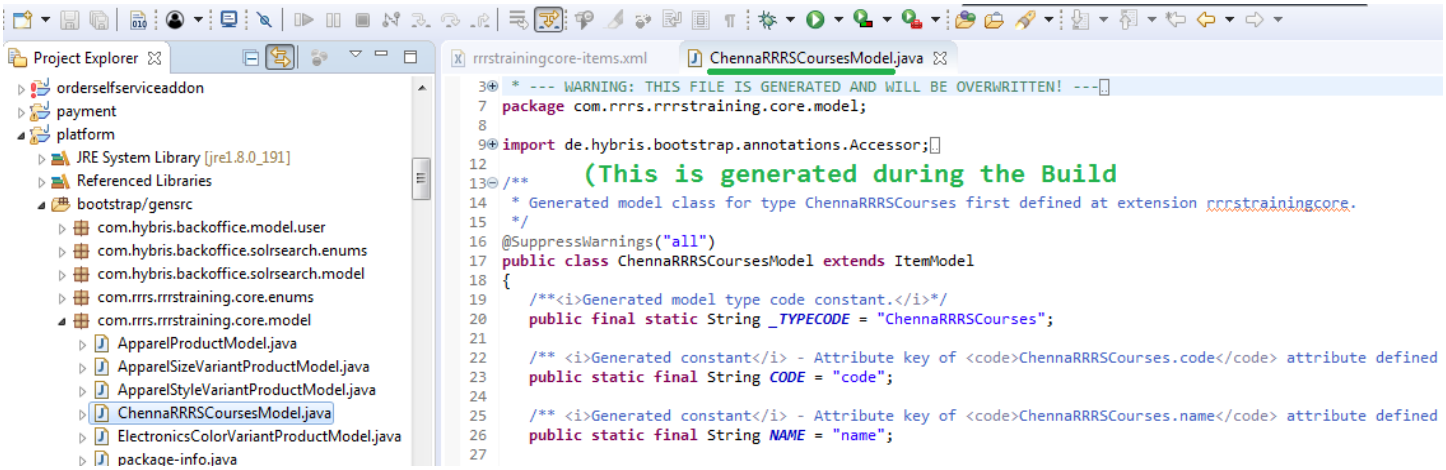


Step 1 = Create itemtype called "**ChennaRRRSCourses**"

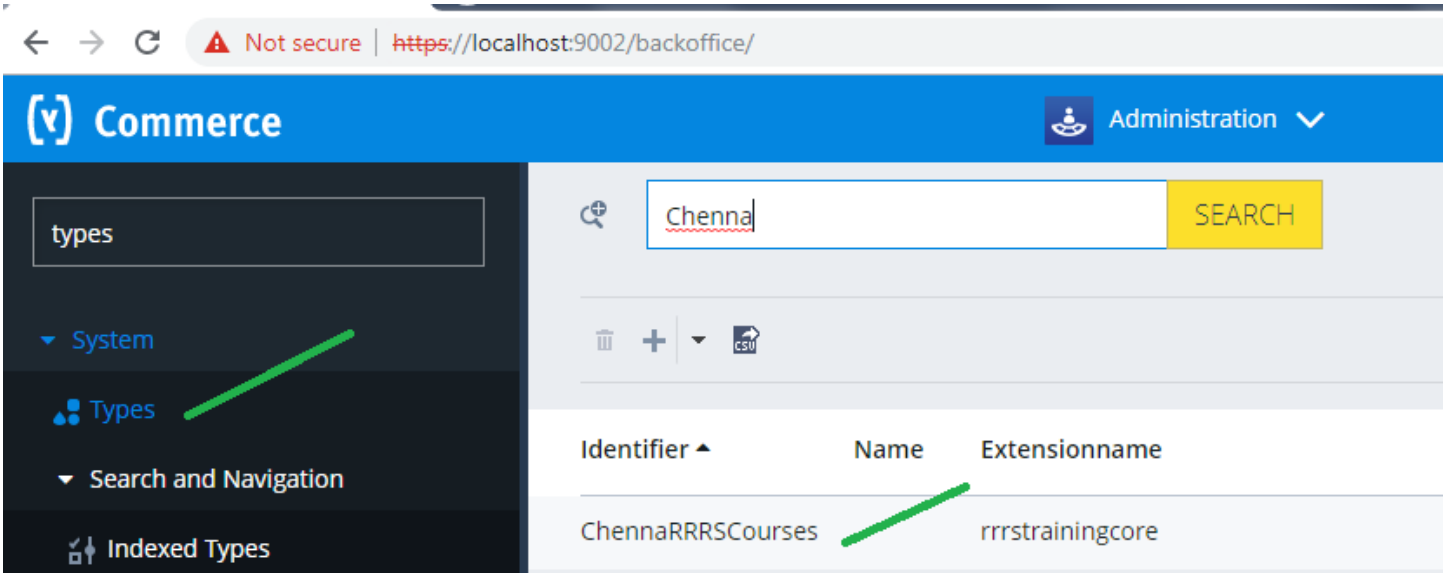


Contact Us = ChennaReddyTraining@RRRS.CO.IN

Step 2 = Build (**ant clean all**) …

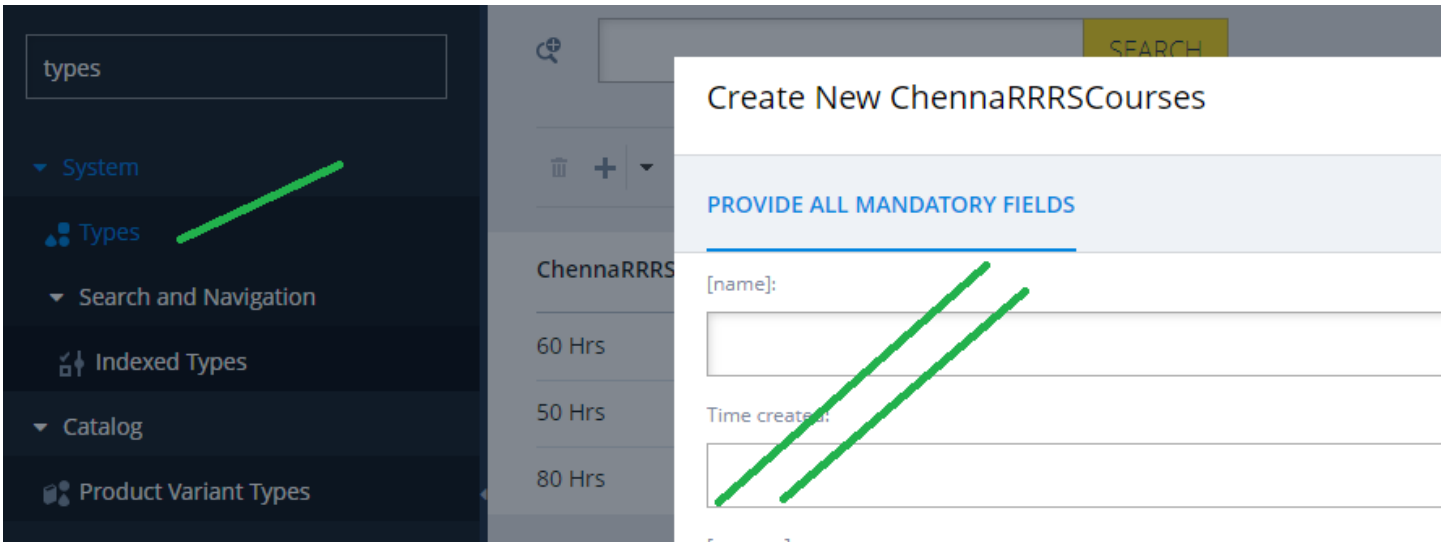


Step 3 = Start the Server (**Hybrisserver.bat**) … hAC – **Update**



Contact Us = ChennaReddyTraining@RRRS.CO.IN

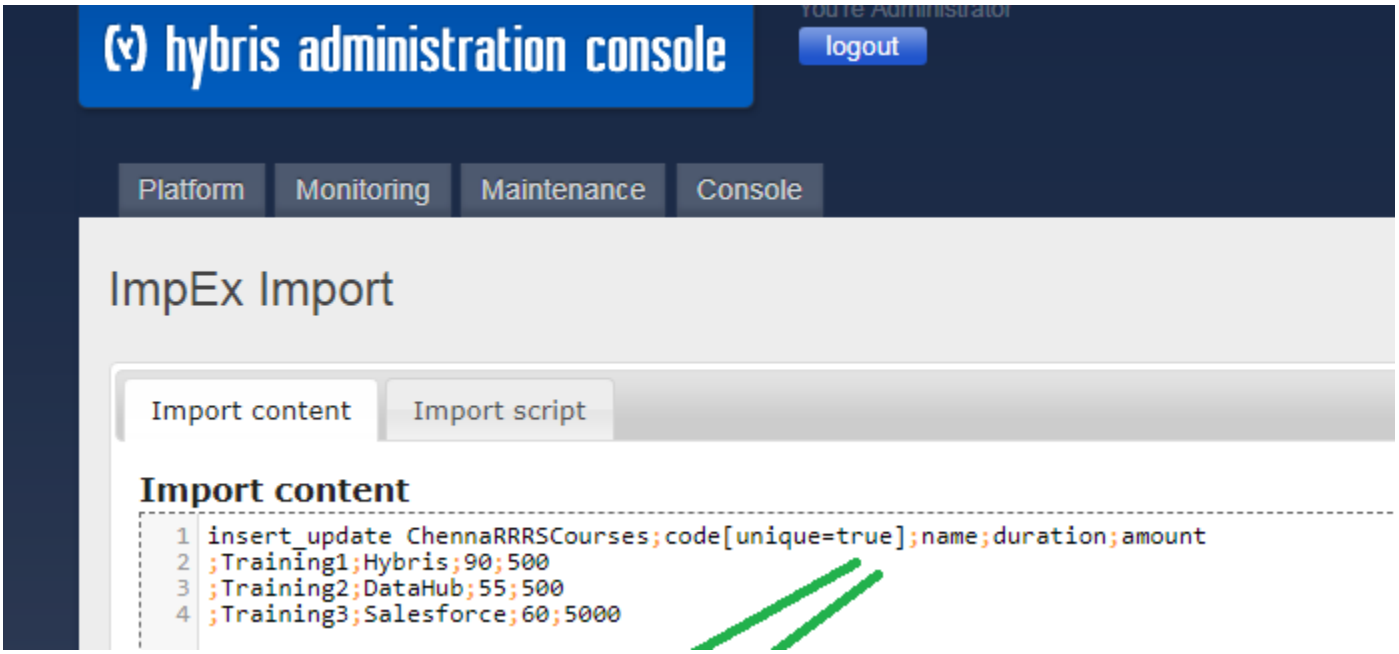
Step 4 = Insert the records into **RRRSTrainingCourses** table.

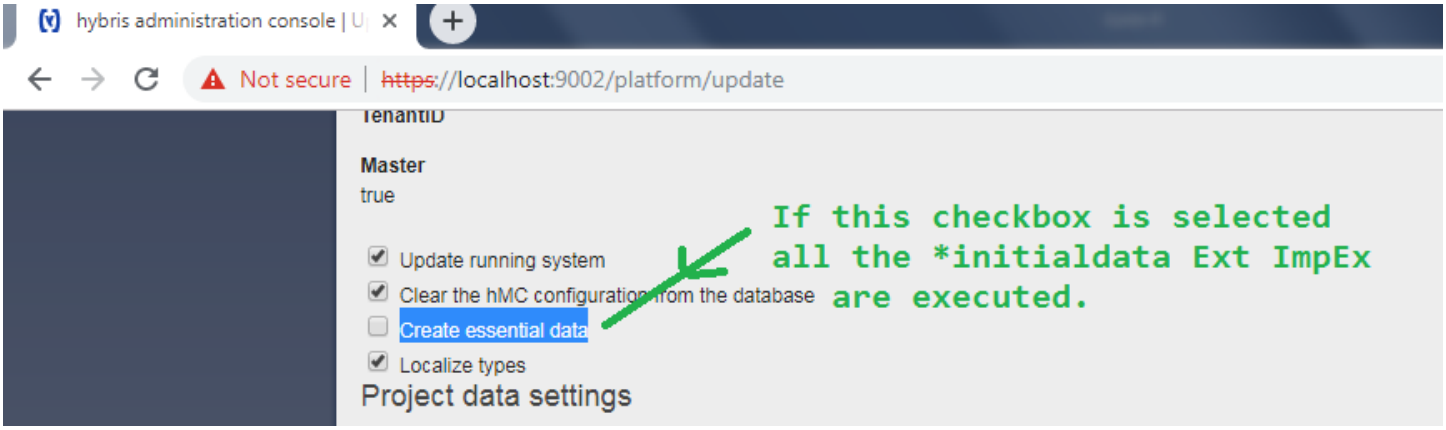


(OR) – **Write ImpEx & Execute from hAC**

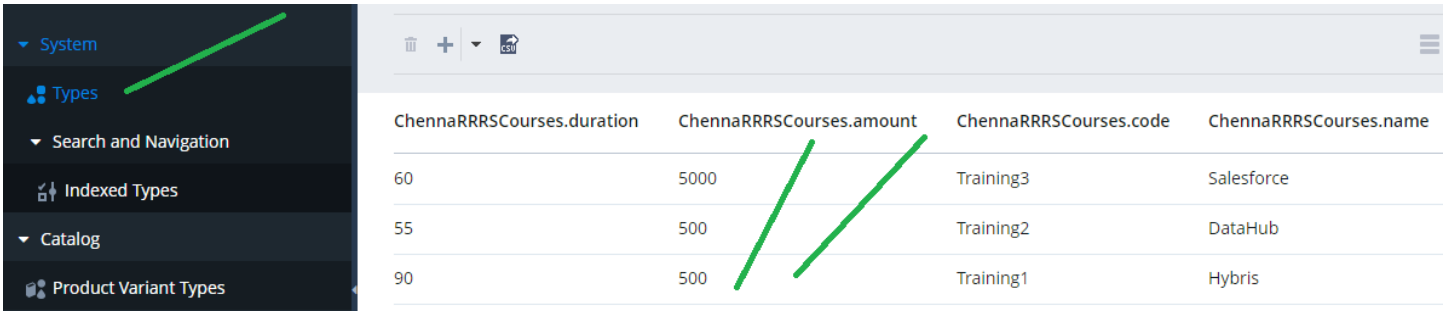
(or) Keep ImpEx in *Initialdata Ext

```
insert_update ChennaRRRSCourses;code[unique=true];name;duration;amount
;Training1;Hybris;90;500
;Training2;DataHub;55;500
;Training3;Salesforce;60;5000
```

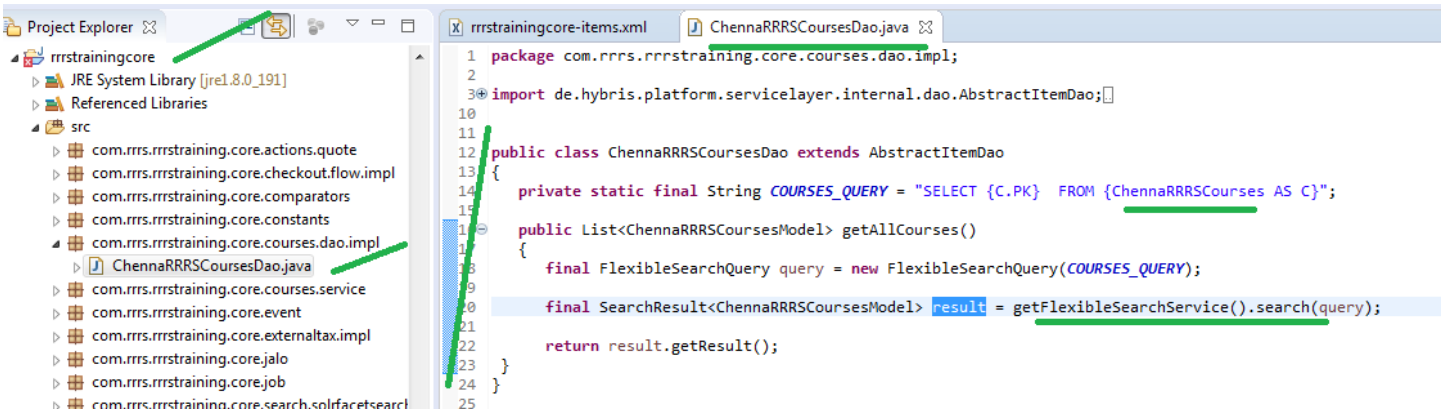


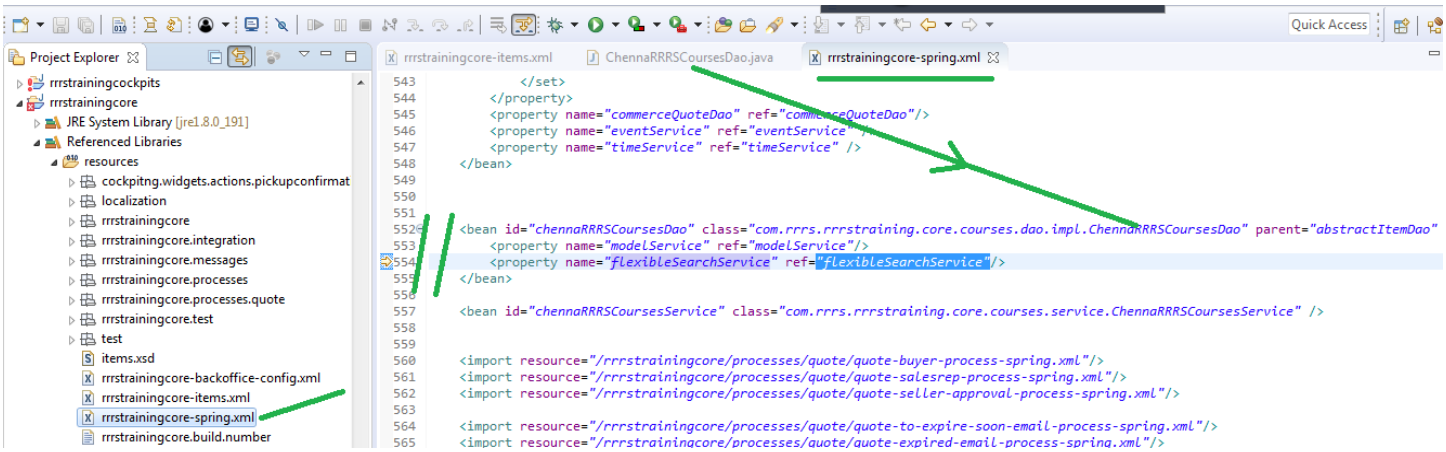


Results =

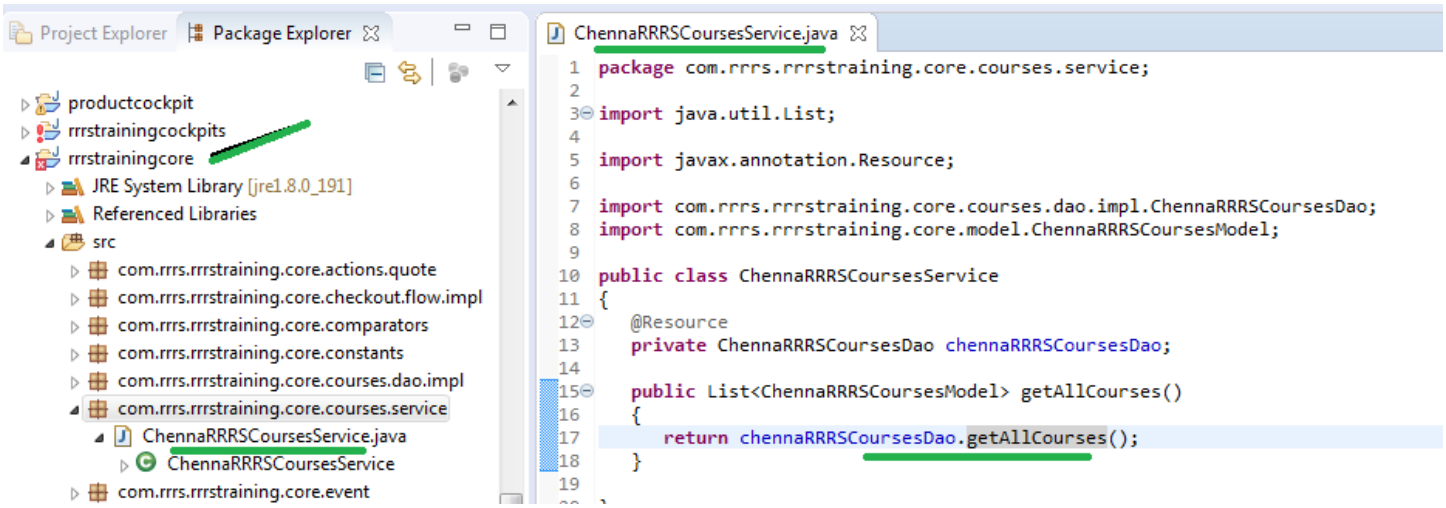


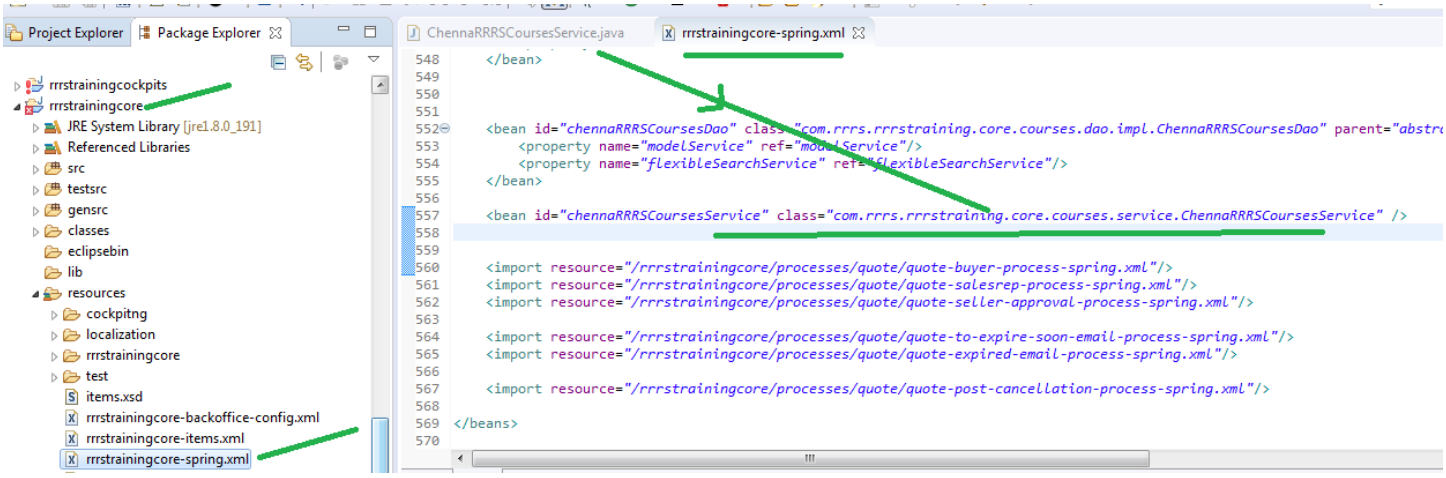
Step 5 = Write Flexible Search Query to **get (or) fetch** the Above table records.





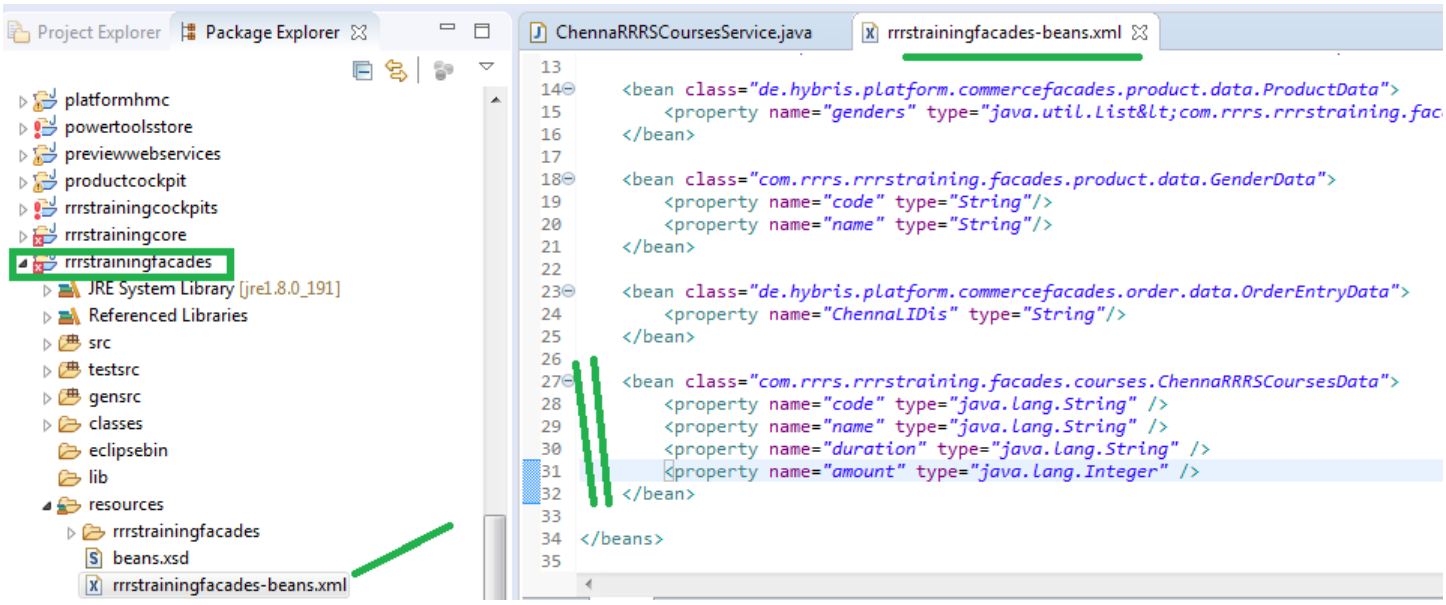
Step 6 = Write Service Layer Class & Call the DAO Class



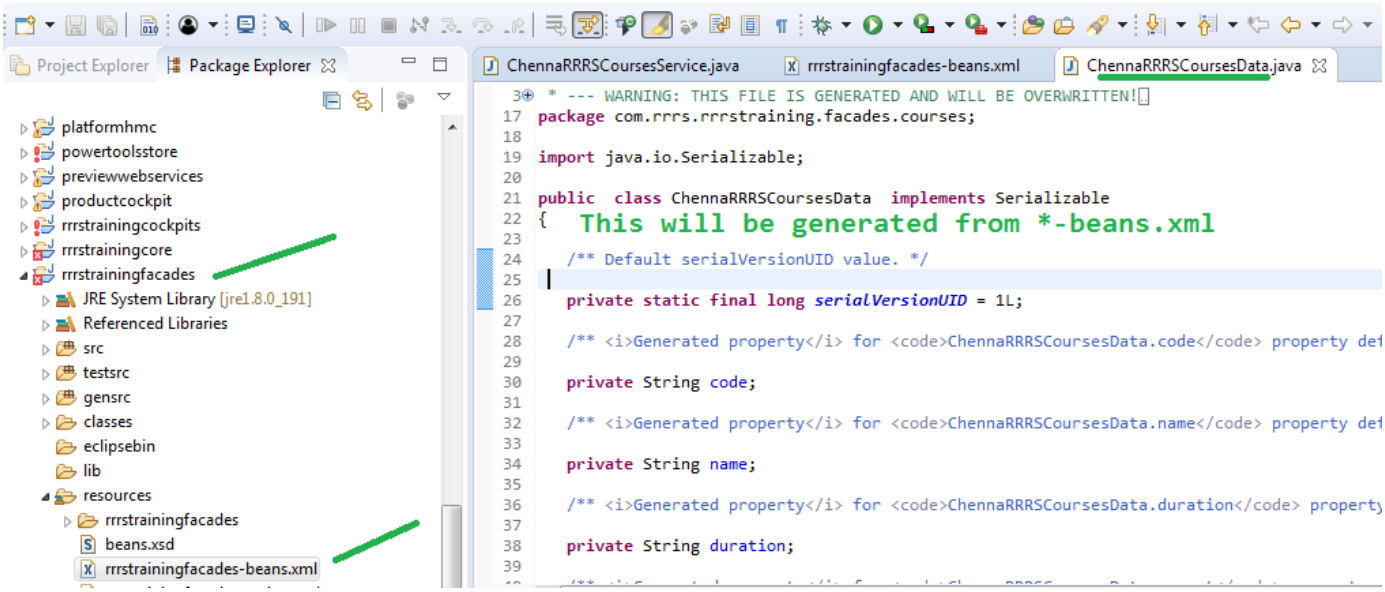


Step 7 = Create the Data Class (or) Data Obj Class

= **ChennaRRRSCoursesData**

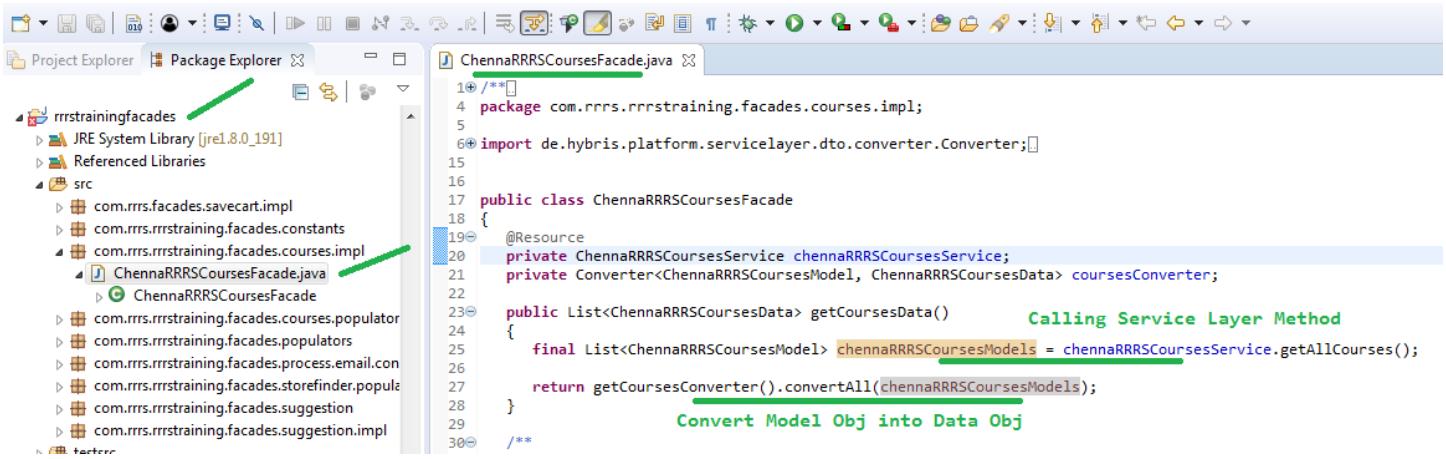


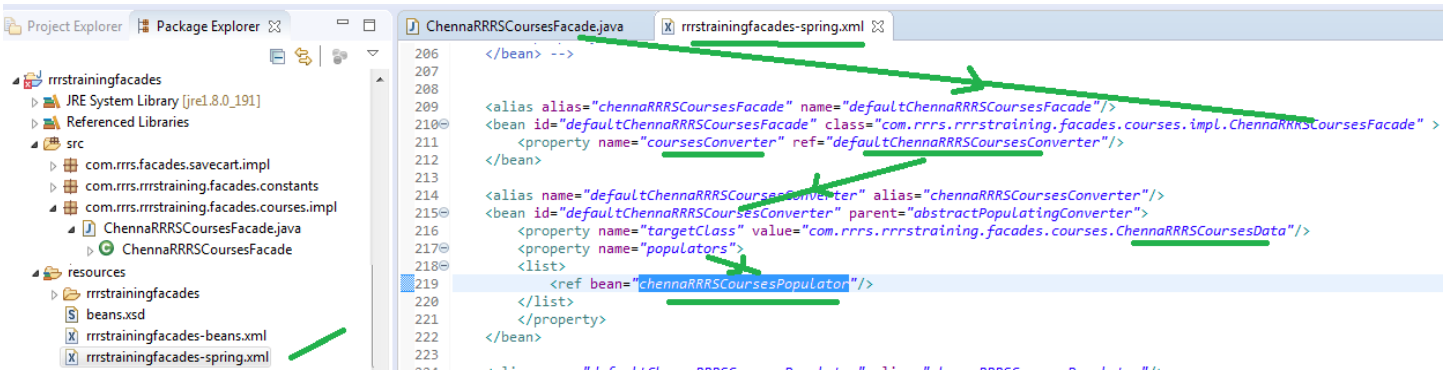
Step 8 = Do the build (**ant clean all**)



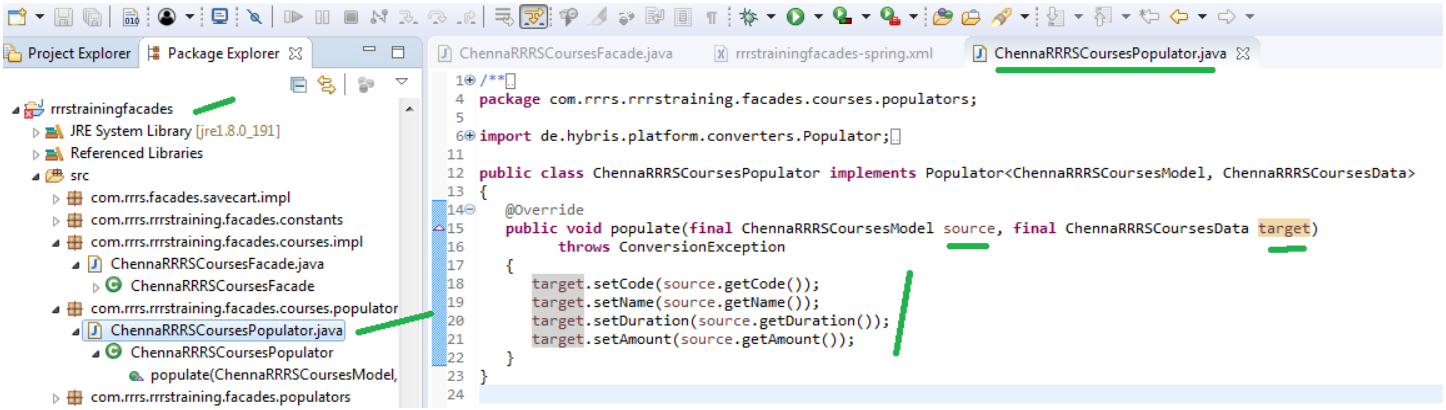
Contact Us = ChennaReddyTraining@RRRS.CO.IN

Step 9 = Write the Façade Class & Service Layer Class





Step 10 = Write the Populator



Contact Us = ChennaReddyTraining@RRRS.CO.IN

Overall Results so far

```
public class ChennaRRRSCoursesDao extends AbstractItemDao
{
   private static final String COURSES_QUERY = "SELECT {C.PK}  FROM {ChennaRRRSCourses AS C}";

   public List<ChennaRRRSCoursesModel> getAllCourses()
   {
      final FlexibleSearchQuery query = new FlexibleSearchQuery(COURSES_QUERY);

      final SearchResult<ChennaRRRSCoursesModel> result = getFlexibleSearchService().search(query);
```

Step 5 = Results

It contains All 3 Records

After Step 8 = Results

ChennaRRRSCoursesData

Code	Name	Duration	Amount
Training1	Hybris	90	500
Training2	Datahub	55	500
Training3	Salesforce	60	500

Step 3 = Results

Table = RRRSTrainingCourses

Code	Name	Duration	Amount
Training1	Hybris	90	500
Training2	Data Hub	55	500
Training3	Salesforce	60	500

Step 4 = Results

Database

Step 10 = Results

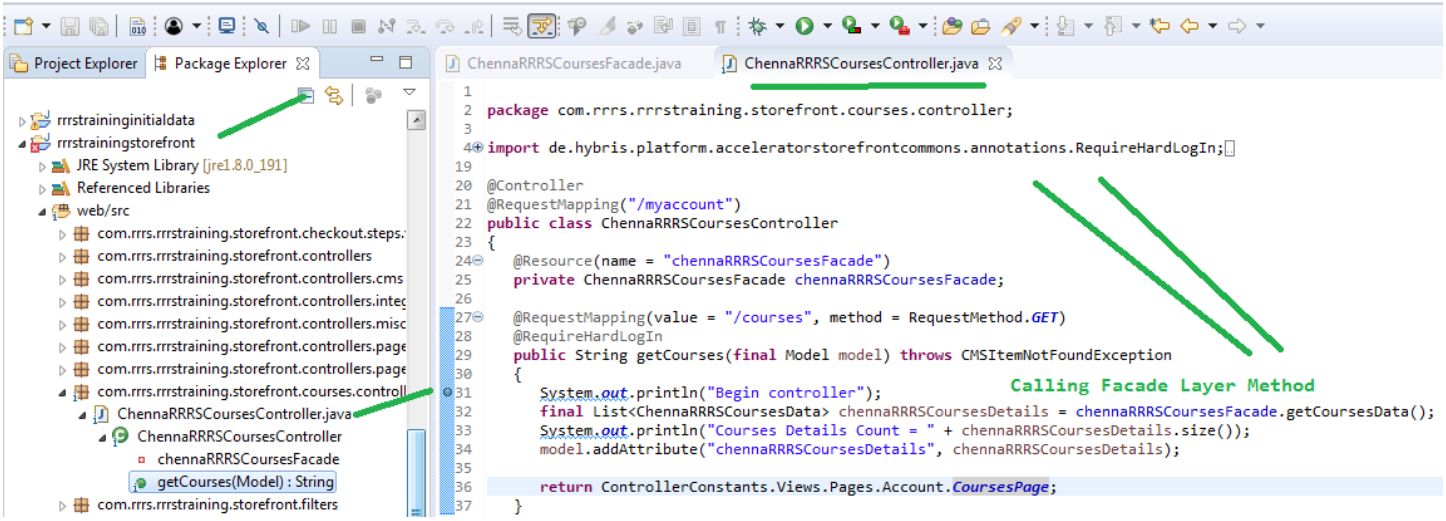
ChennaRRRSCoursesData

Training1	Hybris	90	500
Training2	DataHub	55	500
Training3	Salesfo	60	500

ChennaRRRSCoursesModel

Training1	Hybris	90	500
Training2	Datahub	55	500
Training3	Salesforce	60	500

Step 11 = Write the Controller & Call the Façade Layer

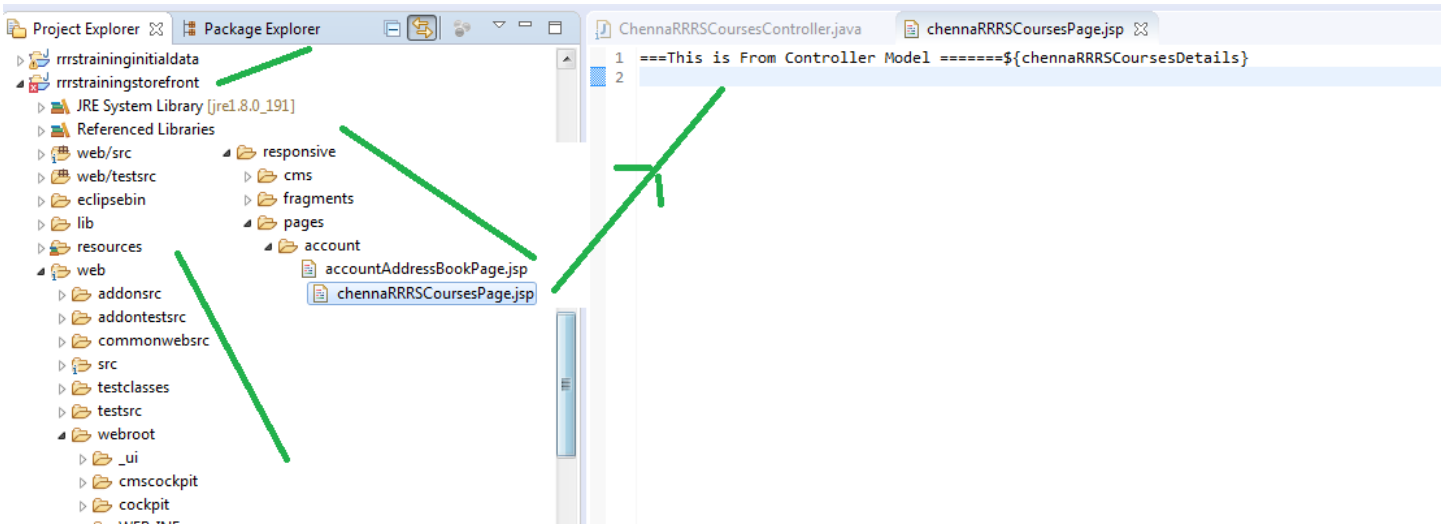


Contact Us = ChennaReddyTraining@RRRS.CO.IN

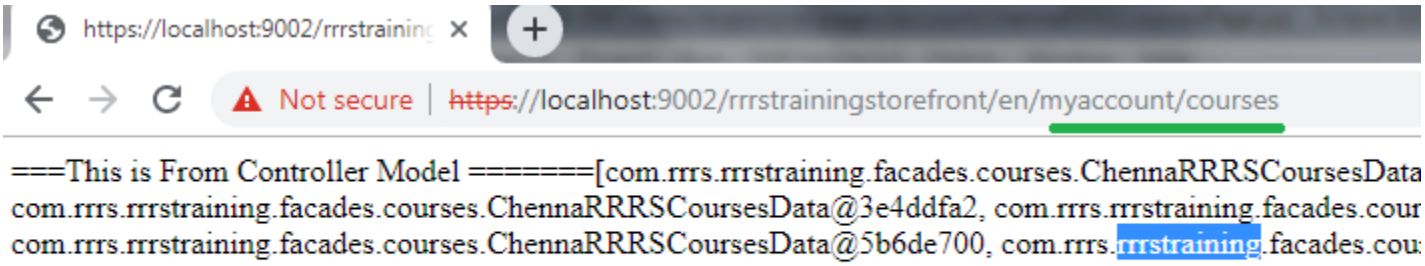
ChennaRRRSCoursesFacade.java | ChennaRRRSCoursesController.java | ControllerConstants.java

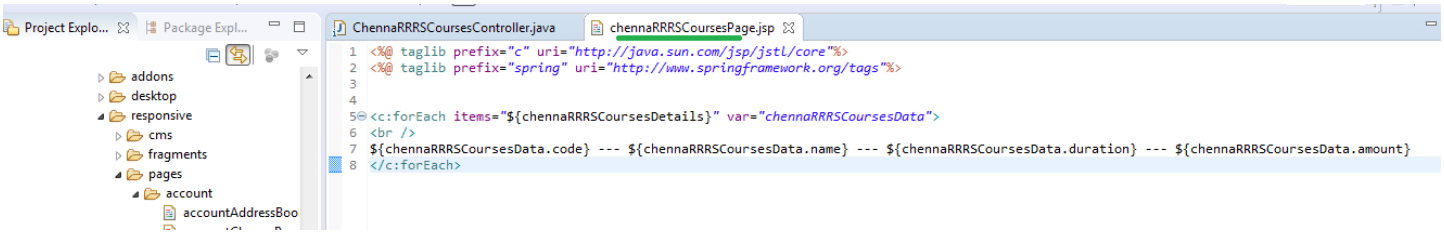
```
88          String AccountProfileEmailEditPage = "pages/account/accountProfileEmailEditPage"; // NOSONAR
89          String AccountChangePasswordPage = "pages/account/accountChangePasswordPage"; // NOSONAR
90          String AccountAddressBookPage = "pages/account/accountAddressBookPage"; // NOSONAR
91          String AccountEditAddressPage = "pages/account/accountEditAddressPage"; // NOSONAR
92          String AccountPaymentInfoPage = "pages/account/accountPaymentInfoPage"; // NOSONAR
93          String AccountRegisterPage = "pages/account/accountRegisterPage"; // NOSONAR
94
95          String CoursesPage = "pages/account/chennaRRRSCoursesPage"; // NOSONAR
96
97      }
98
```

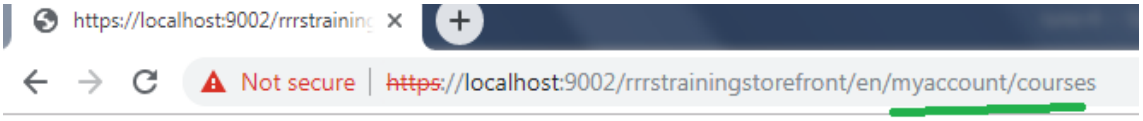
Step 12 = Write the UI Layer (or) Presentation Layer



Results: -







1011 --- Hybris --- 80 Hrs --- 500
1012 --- Data Hub --- 50 Hrs --- 500
1013 --- C Lang --- 60 Hrs --- 500
Training1 --- Hybris --- 90 --- 500
Training2 --- DataHub --- 55 --- 500
Training3 --- Salesforce --- 60 --- 5000

```
public class ChennaRRRSCoursesDao extends AbstractItemDao
{
   private static final String COURSES_QUERY = "SELECT {C.PK}  FROM {ChennaRRRSCourses AS C}";

   public List<ChennaRRRSCoursesModel> getAllCourses()
   {
      final FlexibleSearchQuery query = new FlexibleSearchQuery(COURSES_QUERY);

      final SearchResult<ChennaRRRSCoursesModel> result = getFlexibleSearchService().search(query);
```

Step 3 = Results

Table = RRRSTrainingCourses

Code	Name	Duration	Amount
Training1	Hybris	90	500
Training2	Data Hub	55	500
Training3	Salesforce	60	500

Step 4 = Results

Database

Step 5 = Results
It contains All 3 Records

After Step 8 = Results
ChennaRRRSCoursesData

Code	Name	Duration	Amount
Training1	Hybris	90	500
Training2	Datahub	55	500
Training3	Salesforce	60	500

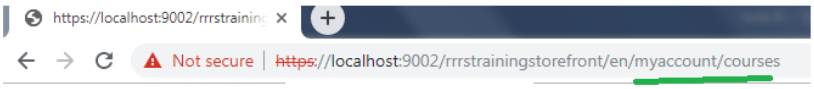
Step 10 = Results

ChennaRRRSCoursesData

Training1	Hybris	90	500
Training2	DataHub	55	500
Training3	Salesfo	60	500

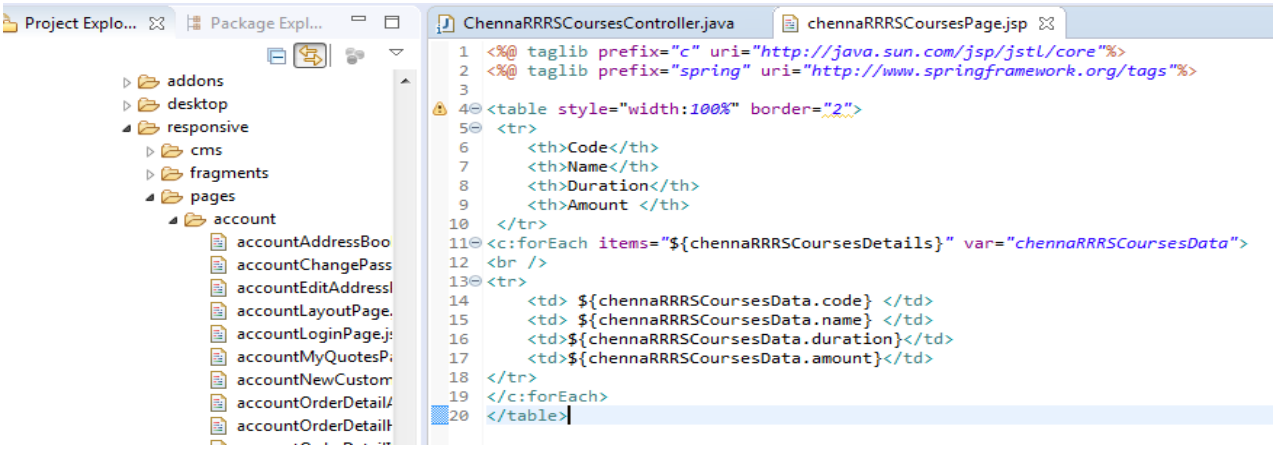
ChennaRRRSCoursesModel

Training1	Hybris	90	500
Training2	Datahub	55	500
Training3	Salesforce	60	500

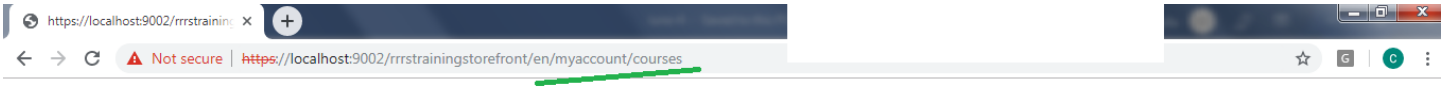


Step 12 = Results

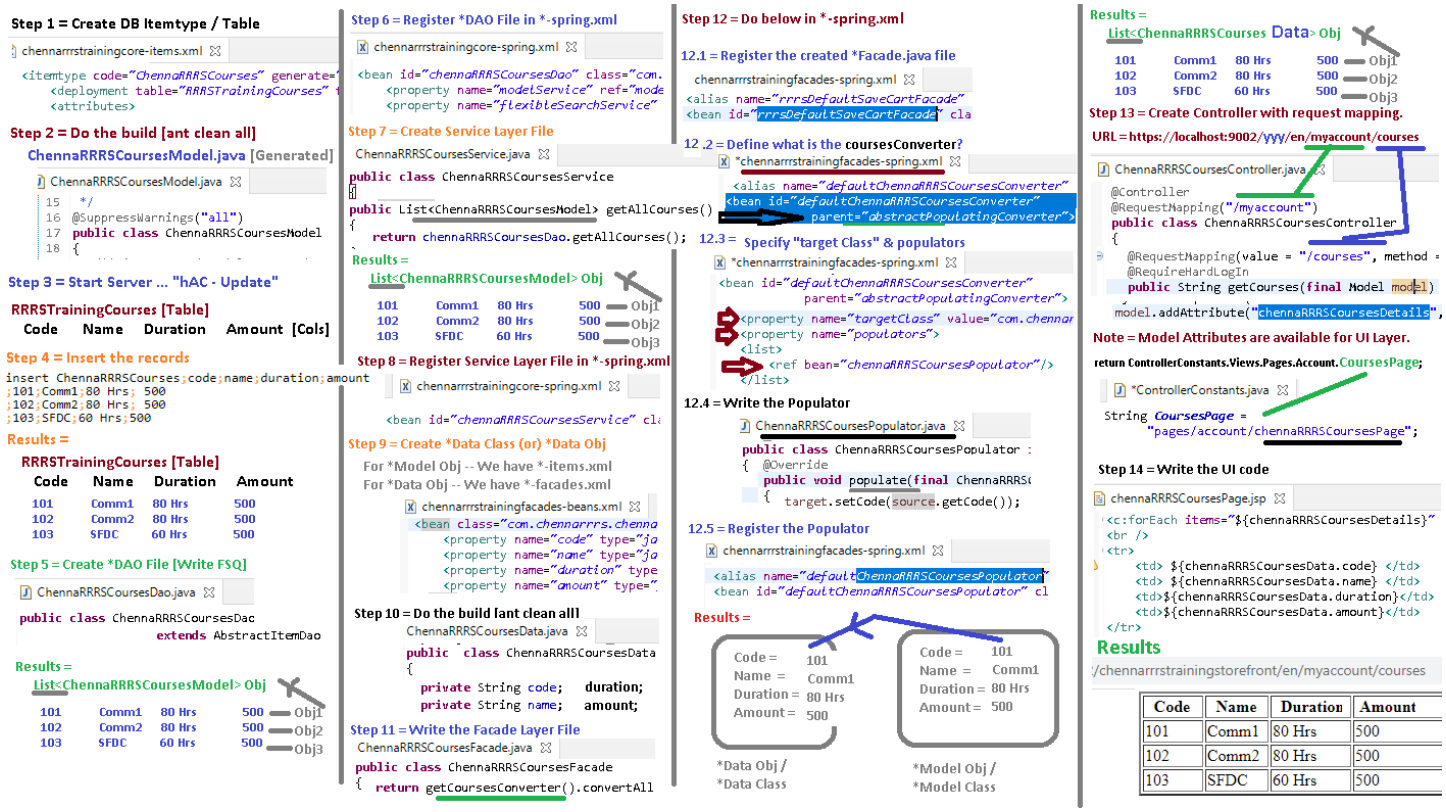
1011 --- Hybris --- 80 Hrs --- 500
1012 --- Data Hub --- 50 Hrs --- 500
1013 --- C Lang --- 60 Hrs --- 500
Training1 --- Hybris --- 90 --- 500



Contact Us = ChennaReddyTraining@RRRS.CO.IN



Code	Name	Duration	Amount
1011	Hybris	80 Hrs	500
1012	Data Hub	50 Hrs	500
1013	C Lang	60 Hrs	500
Training1	Hybris	90	500
Training2	DataHub	55	500
Training3	Salesforce	60	5000



Contact Us = ChennaReddyTraining@RRRS.CO.IN

Interceptors – Models in Hybris (Lifecycle)

Q: What is the purpose of Model classes (or) Model Obj?

Model Obj is the True representation of DB – Table record.

Model in Hybris has lifecycle managed by "**ModelService**".

ModelService – Has the methods to perform CURD Operations.

Create … Update … Load … Remove

Hybris offers Interceptors (Programs) that provides Hook to model lifecycle.

There are 5 types of Interceptors:

a. LoadInterceptor = Invoked whenever a model is loaded from DB

void **onLoad** (Object model, InterceptorContext ctx) throws InterceptorException;

b. InitDefaultsInterceptor = Invoked during modelService.create() & modelService.initDefaults()

void onInitDefaults(Object model, InterceptorContext ctx) throws InterceptorException;

c. PrepareInterceptor = Invoked before model is saved to database & before ValidateInterceptor

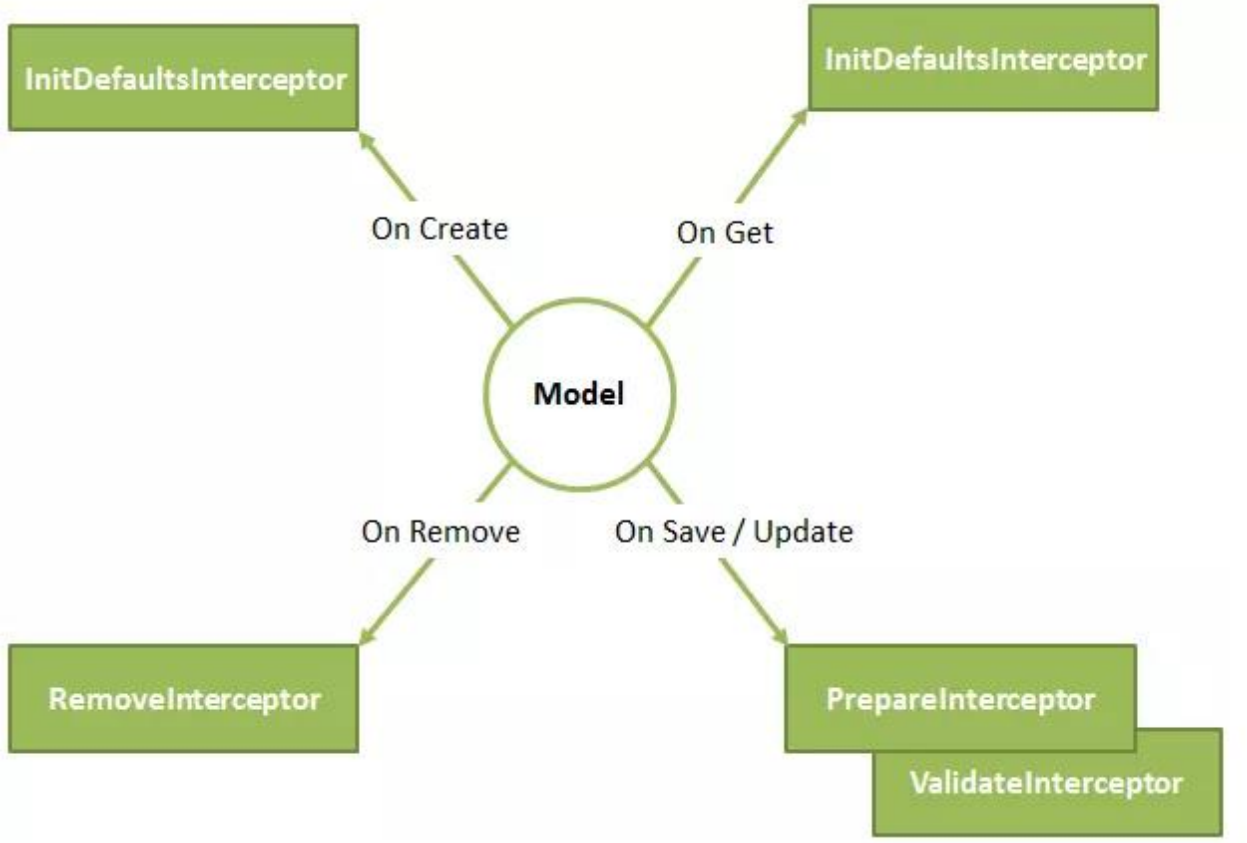
void onPrepare(Object model, InterceptorContext ctx) throws InterceptorException;

d. ValidateInterceptor = Invoked before a model is saved to database & after PrepareInterceptor

void onValidate(Object model, InterceptorContext ctx) throws InterceptorException;

e. RemoveInterceptor = Invoked before a model is removed from the DB.

void onRemove(Object model, InterceptorContext ctx) throws InterceptorException;



These models we use in Hybris has its own lifecycle managed by **ModelService**.

There are several methods provided by ModelService to represent its lifecycle.

Example: Create, Save, Update, Load, Remove.

When we call any of these ModelService method, it just follows its Lifecycle and completes the execution.

Example: Save method save the model data in Database.

Q = Why we need interceptors?

The interceptors have the ability to interrupt the **lifecycle of a model** and execute some code and then allow the model to continue its lifecycle.

Q = When we use interceptors?

We need to use interceptors whenever we want to perform some action during the life cycle of a model.

> **Example:** If we need to validate the model data before saving in database

We can use interceptors in some scenarios as mentioned below: -

> **1.** Auditing the details while saving the model data
>
> **2.** Validating the model data before saving
>
> **3.** Restrict data from deleting or modifying based on some conditions

We can even modify the model before saving it using interceptors.

Note = Each of above interceptor addresses a particular step of the model life cycle.

When the life cycle of a model reaches a certain step, a corresponding interceptor will be executed automatically.

Note = Remove interceptor is represented by interface called "**RemoveInterceptor**"

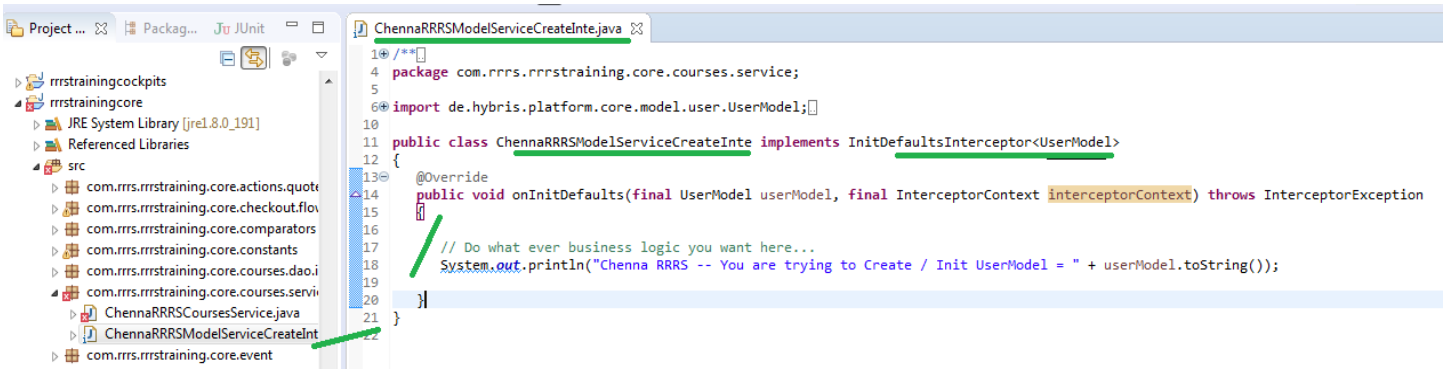
This is called before a model is **removed/deleted** from the database.

We can use remove interceptor whenever we need to remove models that are related to the model but are not in the model context.

Example for “Create – InitDefaultsInterceptor”

Step 1 = Create a class “**ChennaRRRSModelServiceCreateInte**” & implement interface called “**InitDefaultsInterceptor**”.

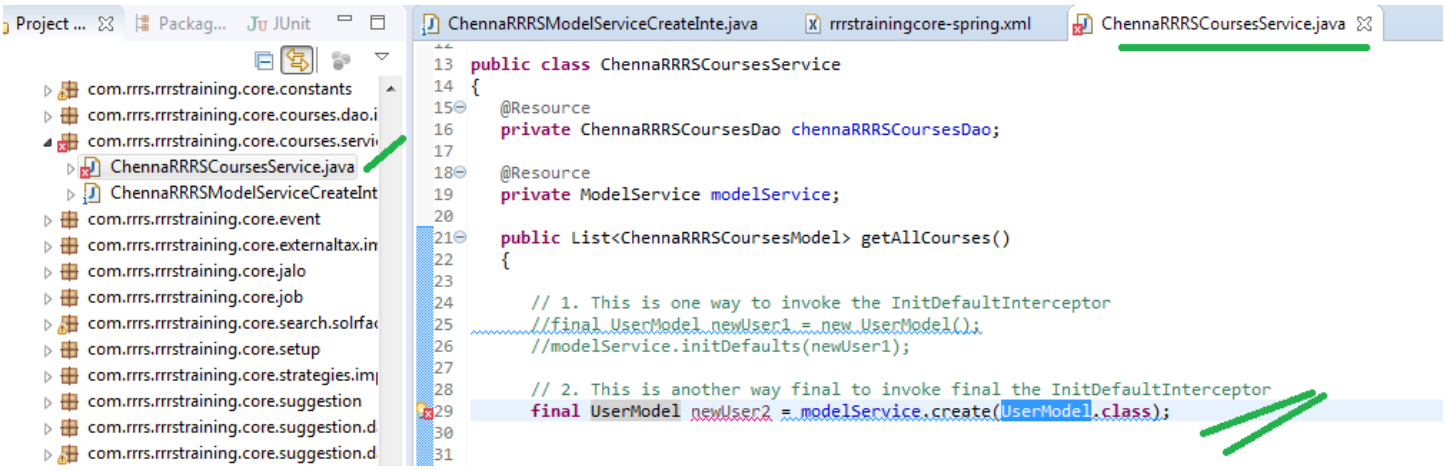
InitDefaultsInterceptor – This interface having the methods & override method called “onInitDefaults()”



Step 2 = Register “**ChennaRRRSModelServiceCreateInte**” as a bean in ***-spring.xml**

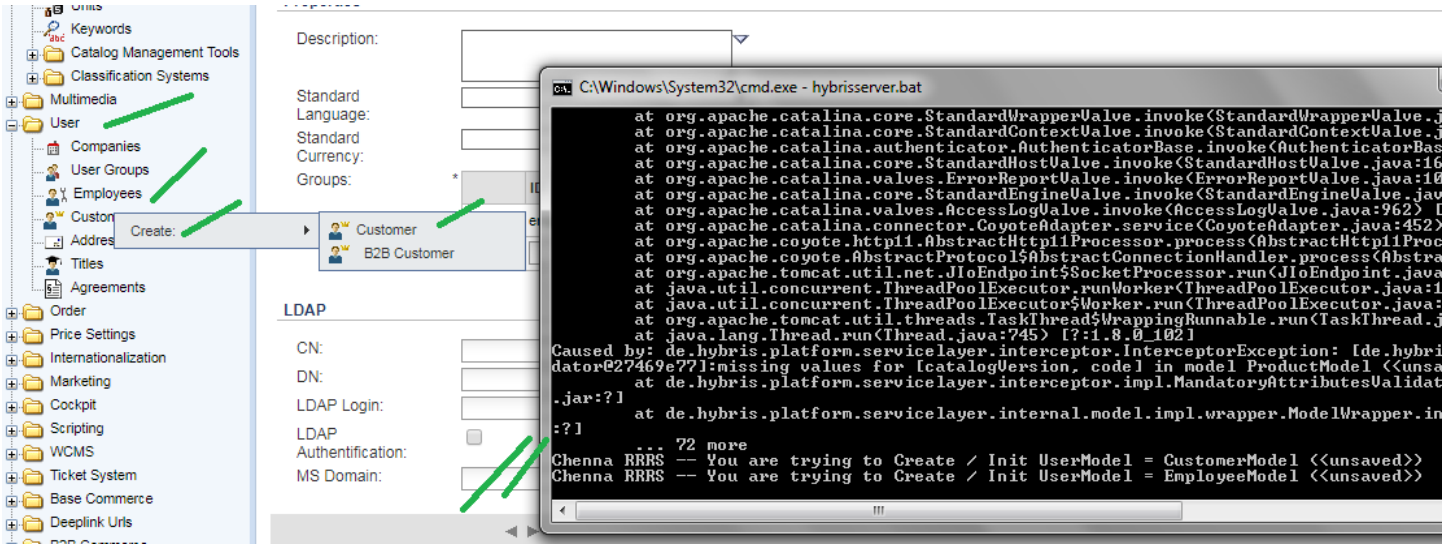
Step 3 = Declare & Configure your interceptor using “InterceptorMapping”.

Step 4 = In any class … Whenever you create “Use Model” instance then you can see “Our Interceptor” will be executed.

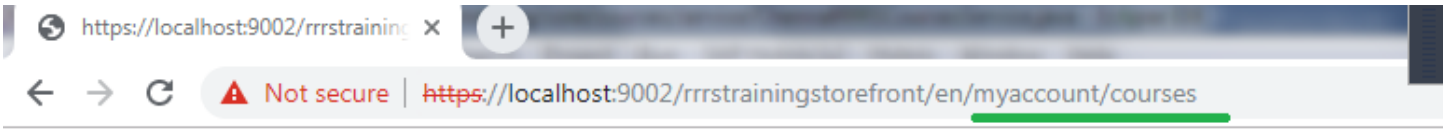


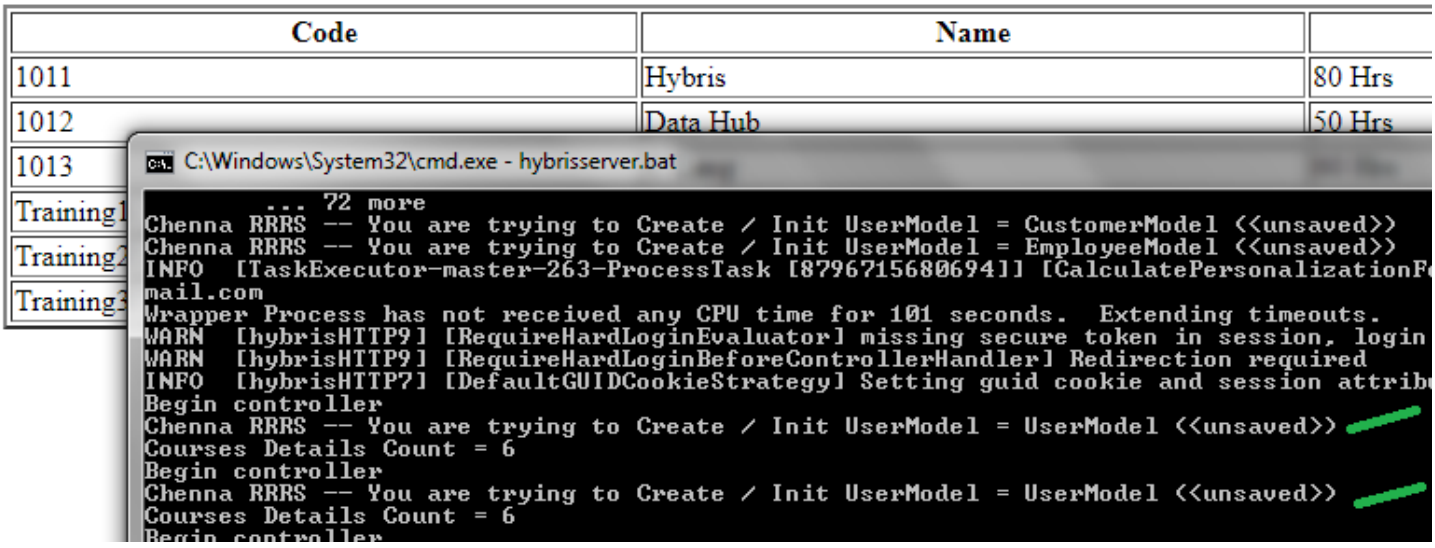
Step 5 = Test the Results: -

hMC / Backoffice: -



(or)





Note: - LoadInterceptor this interface is having “onload()” method override it. modelService.get(XXX) calls above override method.