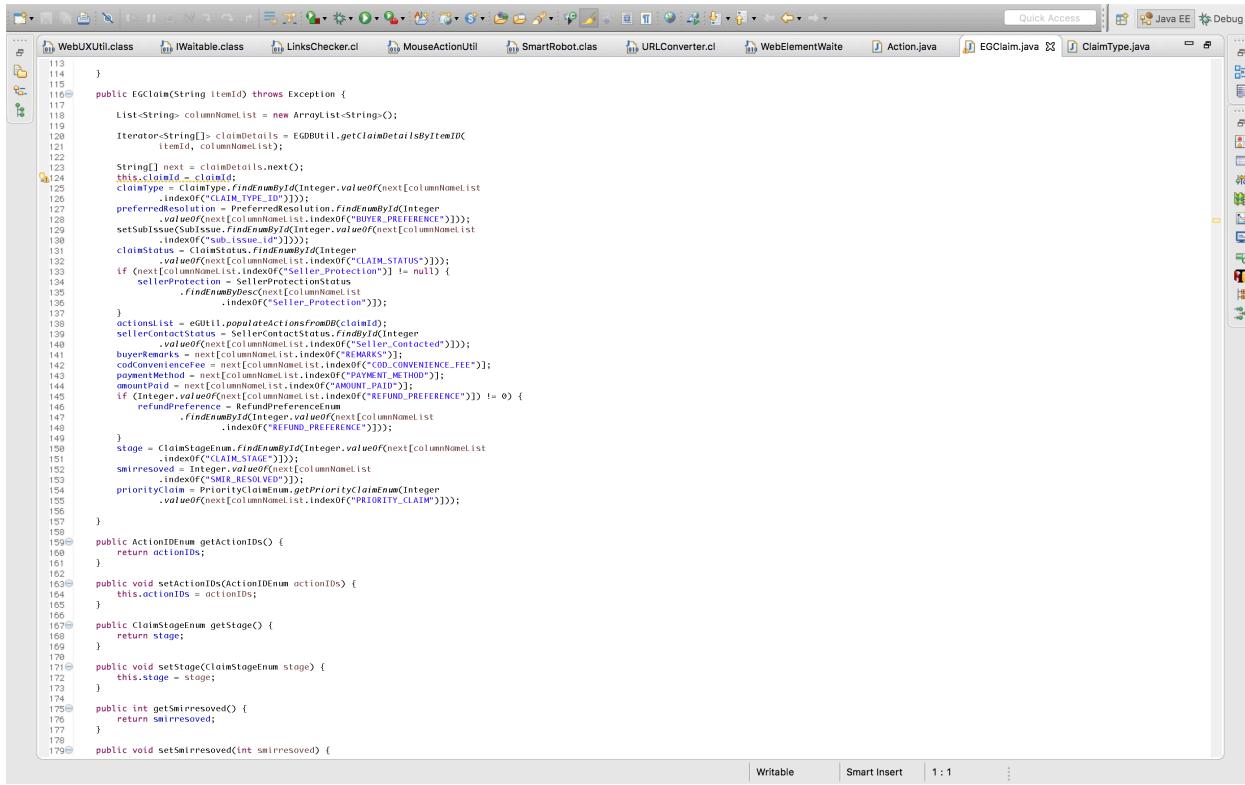


Screenshot of Eclipse IDE showing the Java EE perspective. The left side shows the Project Explorer with numerous Java files under com.ebay.mauli.component.ebayIn.eG.common. The center shows the code editor with URLConverter.cl, which includes imports for com.ebay.mauli.component.billing.payments.ebay.utils.api.pgwr.request, com.ebay.mauli.component.billing.payments.ebay.utils.soa, com.ebay.mauli.component.billing.payments.ebay.utils.soa.billng, com.ebay.mauli.component.billing.payments.ebay.utils.soa.pgwservi, and com.ebay.mauli.component.billing.payments.ebay.utils.soa.pgwserv. The code defines a public enum ClaimType with various items like ITEM_NOT_RECEIVED, ITEM_NOT_AS_DESCRIBED, ITEM_NOT_WORKING_BROKEN, PART_OF_THE_ORDER_MISSING, PART_OF_PRODUCT_MISSING, DAMAGED_ITEM, INVOICE_ISSUE, EMPTY_PARCEL_RECEIVED, and Empty parcel received. It also includes methods for getting the claim type by id and description. The right side shows the Type Hierarchy view for the ClaimType enum, listing its members and their descriptions.

Screenshot of Eclipse IDE showing the Java EE perspective. The left side shows the Project Explorer with the same set of Java files as the first screenshot. The center shows the code editor with URLConverter.cl, which now includes imports for java.util.ArrayList and java.util.List. The class EGClaim is defined with many private fields: claimId, claimType, subIssue, subIssueList, preferredResolution, claimStatus, claimStatusList, sellerProtection, actionsList, sellerContactStatus, buyerRemarks, egShipmentList, claimCreationDate, lastModifiedDate, claimCreatedBy, claimOpeningDate, claimClosingDate, sellerProtectionReopen, dateFilter, egCSVParam, paymentMethod, codConvenienceFee, orderValueFee, amountPaid, disputeChannel, claimClosedBy, itemPrice, actionDesc, resolutionSummary, stage, isSatisfied, ActionIDEnum, actionIDs, refundPreferenceenum, refundPreference, priorityClaimNum, priorityClaim, isImageDicsChecked, isImageDownloaded, isImageMandatory, beneficiaryId, beneficiaryAddress, isSRNNeed0IP, resendOTP, alreadyOTPVerified, and a constructor EGClaim(). The right side shows the Type Hierarchy view for the EGClaim class, listing its properties and their types.



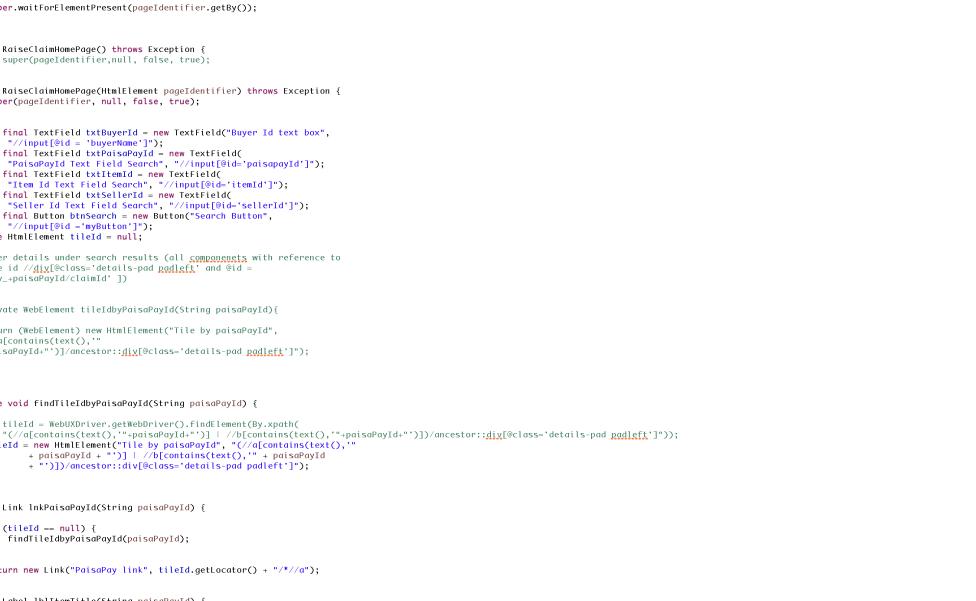
The screenshot shows a Java IDE interface with the following details:

- Title Bar:** Quick Access, Java EE, Debug.
- Project Explorer:** Shows files like WebUXUtil.class, IWaitable.class, LinksChecker.cl, MouseActionUtil, SmartRobot.cl, URLConverter.cl, WebElementWaite, Action.java, EGClaim.java (selected), and ClaimType.java.
- Code Editor:** Displays the content of the EGClaim.java file. The code is a complex class with many methods and nested blocks, primarily dealing with database queries and object manipulation.
- Status Bar:** Shows Writable, Smart Insert, and a zoom ratio of 1:1.

```
113     }
114 }
115 public EGClaim(String itemId) throws Exception {
116     List<String> columnNamesList = new ArrayList<String>();
117     Iterator<String> columnIterator = EGBBUtil.getClaimDetailsByItemId(
118         itemId, columnNamesList);
119     String[] next = columnIterator.next();
120     this.claimId = claimId;
121     claimType = claimType.findByNameById(Integer.valueOf(next[columnNamesList
122         .indexOf("CLAIM_TYPE_ID")]));
123     preferredSeller = sellerContractStatus.findByNameById(Integer
124         .valueOf(next[columnNamesList.indexOf("BUYER_PREFERENCE"))));
125     setSubIssue(subIssue.findByNameById(Integer.valueOf(next[columnNamesList
126         .indexOf("SUB_ISSUE_ID"))));
127     claimStatus = claimStatus.findByNameById(Integer
128         .valueOf(next[columnNamesList.indexOf("CLAIM_STATUS"))));
129     if (next[columnNamesList.indexOf("Seller_Protection")] != null) {
130         sellerProtection = SellerProtectionStatus
131             .findByName(next[columnNamesList
132                 .indexOf("Seller_Protection")));
133     }
134     actionsList = EGBUtil.getActionsFromDb(itemId);
135     sellerContractStatus = sellerContractStatus.findByNameById(
136         Integer.valueOf(next[columnNamesList.indexOf("Seller_Contacted"))));
137     buyerRemarks = next[columnNamesList.indexOf("REMARKS")];
138     convenienceFee = next[columnNamesList.indexOf("CONVENIENCE_FEE")];
139     paymentMethod = next[columnNamesList.indexOf("PAYMENT_METHOD")];
140     amountPaid = next[columnNamesList.indexOf("AMOUNT_PAID")];
141     if (Integer.valueOf(next[columnNamesList.indexOf("REFUND_PREFERENCE"))] != 0) {
142         refundPreference = RefundPreference
143             .findByName(next[columnNamesList
144                 .indexOf("REFUND_PREFERENCE")));
145     }
146     stage = ClaimStageEnum.findByNameById(Integer.valueOf(next[columnNamesList
147         .indexOf("CLAIM_STAGE"))));
148     smirresolved = Integer.valueOf(next[columnNamesList
149         .indexOf("SMIR_RESOLVED")));
150     priorityClaim = priorityClaimNum.getPriorityClaimEnum(Integer
151         .valueOf(next[columnNamesList.indexOf("PRIORITY_CLAIM"))));
152 }
153
154 public ActionIDEnum getActionIDs() {
155     return actionIDs;
156 }
157
158 public void setActionIDs(ActionIDEnum actionIDs) {
159     this.actionIDs = actionIDs;
160 }
161
162 public ClaimStageEnum getStage() {
163     return stage;
164 }
165
166 public void setStage(ClaimStageEnum stage) {
167     this.stage = stage;
168 }
169
170 public int getSmirresolved() {
171     return smirresolved;
172 }
173
174 public void setSmirresolved(int smirresolved) {
175 }
```

The screenshot shows the Eclipse IDE interface with several open tabs and panes. The left pane, 'Project Explorer', lists numerous Java files under two main packages: 'com.ebay.maul.component.ebayn.eG.pages' and 'com.ebay.maul.component.ebayn.eG.servicePages'. The right pane displays the source code for the file 'RaiseClaimHomePage.java'. The code is annotated with Javadoc-style comments and uses Selenium WebDriver annotations like @PageIdentifier and @FindBy. The code handles user login, navigating through various claim status pages, and interacting with dropdown menus and buttons. The bottom center of the screen shows a message: 'No operations to display at this time.'

```
1 package com.ebay.maul.component.ebayn.eG.pages;
2
3 import org.openqa.selenium.*;
4
5 public class RaiseClaimHomePage extends WebPage {
6
7     private static final String URL = "String" ContextManager
8         .getThreadContext().getAttribute("eGUrl");
9
10    private static final HtmlElement pageIdentifier = new TextField(
11        "BuyerId search Text box", "#input19id = 'buyerName'");
12    private static final HtmlElement buyerName = new HtmlElement(
13        "Buyer Name", "#div#id = 'buyerNameDiv'") HtmlElement
14
15    private static final HtmlElement blockUIBlock = new HtmlElement(
16        "UI blocker during search", "//div[@class = 'blockUI blockOverlay']");
17
18    private final Link InvViewShippingLabel = new Link(
19        "View Shipping Label",
20        "//a[contains(text(), 'View Shipping Label') or contains(text(), 'View Shipping Label')]");
21
22    private final Link InvViewShippingDeclarationLabel = new Link(
23        "View Shipping Declaration Label",
24        "//a[contains(text(), 'View Shipping Declaration')]");
25
26    /**
27     * @param userName
28     * @param password
29     * @param role
30     * @throws Exception
31     */
32    public RaiseClaimHomePage(String userName, String password)
33        throws Exception {
34        super(URL);
35        SignInPage.signIn(userName, password);
36        new WebPage(URL, false);
37        super.waitForElementPresent(pageIdentifier.get());
38    }
39
40    /**
41     * Common Constructor to Access EG homepage. Pass username and
42     * password as null if CSR1 login
43     */
44
45    public RaiseClaimHomePage() {
46        super(URL);
47        SignInPage.signIn(userName, password);
48        new WebPage(URL, false);
49        super.waitForElementPresent(pageIdentifier.get());
50    }
51
52
53    private final Button remindMeLater = new Button("May Be Later Button",
54        "//a[contains(text(), 'Remind me later')]");
55
56    private final Button mayBeLater = new Button("Remind me later Button",
57        "//a[contains(text(), 'Maybe later')]");
58
59    /**
60     * @param userName
61     * @param password
62     */
63    public RaiseClaimHomePage(String userName, String password,
64        HtmlElement pageIdentifier) throws Exception {
65        super(URL, false);
66        SignInPage.signIn(userName, password);
67        new WebPage(URL, false);
68
69        /*try {
70
71        */
72
73        /*
74        */
75
76        /*
77        */
78
79        /*
80        */
81
82        /*
83        */
84
85        /*
86        */
87
88        /*
89        */
90
91        /*
92        */
93
94        /*
95        */
96
97        /*
98        */
99
100    }
101}
```



The screenshot shows the Eclipse IDE interface with the following details:

- Title Bar:** Shows multiple open files including "LinksChecker.cl", "MouseActionUtil", "SmartRobot.clas", "URLConverter.cl", "WebElementWaite", "Action.java", "EGClaim.java", "ClaimType.java", "RaiseClaimHomeP", and "RaiseClaimHomeP".
- Toolbar:** Standard Eclipse toolbar with icons for file operations, search, and preferences.
- Left Sidebar:** Shows the package explorer with several Java packages and files listed.
- Right Sidebar:** Shows the Java EE perspective with various icons for deployment, server management, and resource configuration.
- Code Editor:** The main area displays Java code for a class named "RaiseClaimHomePage". The code includes imports for `java.util.List`, `java.util.ArrayList`, `java.util.Map`, `java.util.HashMap`, `org.openqa.selenium.By`, `org.openqa.selenium.WebDriver`, `org.openqa.selenium.WebElement`, `org.openqa.selenium.support.ui.ExpectedConditions`, `org.openqa.selenium.support.ui.WebDriverWait`, and `org.openqa.selenium.support.ui.FluentWait`.
- Code Content:** The code defines several methods:
 - `public void clickBuyerId()`
 - `public void clickPoisoPayId()`
 - `public void clickSearch()`
 - `private void findTitleIdByPoisoPayId(String poisoPayId)`: This method uses FluentWait to find an element with id "titleId" containing the text "Buyer Id". It then finds the first child element with id "poisoPayId" and returns its locator.
 - `public Link linkPoisoPayId(String poisoPayId)`: Returns a new `Link` object with the href attribute set to the locator of the element found by `findTitleIdByPoisoPayId`.
 - `public Label labelTitleId(String poisoPayId)`: Returns a new `Label` object with the text "Item title" and the locator of the element found by `findTitleIdByPoisoPayId`.
- Status Bar:** Shows "Writable" and "Smart Insert" status, and a vertical position indicator at "1:1".

The screenshot shows the Eclipse IDE interface with the Java EE perspective selected. The top menu bar includes 'File', 'Edit', 'Select', 'Run', 'View', 'Search', 'Help', and 'Quick Access'. The title bar displays the current file as 'LinksChecker.cl'. The left side features a 'Project Explorer' view with icons for files, folders, and other project components. The main workspace contains the Java code for the 'LinksChecker' class, which includes methods for finding various labels and buttons based on their locators and spans.

```
143     return new Label("Item title", tileId.getLocator()
144         + "/*//div[@class='title_padding']/span/b");
145     }
146 }
147
148 public Label lblTxnDate(String paisaPayId) {
149
150     if (tileId == null) {
151         findTileIdByPaisaPayId(paisaPayId);
152     }
153
154     return new Label("Transaction date", tileId.getLocator()
155         + "/*//li[2]/span");
156 }
157
158 public Label lblItemId(String paisaPayId) {
159
160     if (tileId == null) {
161         findTileIdByPaisaPayId(paisaPayId);
162     }
163
164     return new Label("Item Id", tileId.getLocator() + "/*//li[3]/span");
165 }
166
167 public Label lblSellerId(String paisaPayId) {
168
169     if (tileId == null) {
170         findTileIdByPaisaPayId(paisaPayId);
171     }
172
173     return new Label("Seller Id", tileId.getLocator() + "/*//li[4]/span");
174 }
175
176 public Label lblBuyerId(String paisaPayId) {
177
178     if (tileId == null) {
179         findTileIdByPaisaPayId(paisaPayId);
180     }
181
182     return new Label("Buyer Id", tileId.getLocator() + "/*//li[5]/span");
183 }
184
185 public Button btnSeeDetails(String paisaPayId) {
186
187     if (tileId == null) {
188         findTileIdByPaisaPayId(paisaPayId);
189     }
190
191     return new Button("See details button", tileId.getLocator()
192         + "/*//input[contains(@id,'myButton')]");
193 }
194
195 public void clickBtnSeeDetailsByClaimID(String claimID) {
196
197     this.driver.findElement(By.id("myButton_" + claimID)).click();
198 }
199
200
201 public void searchByPaisaPayID(String paisaPayId) {
202
203     txtPaisaPayId.type(paisaPayId);
204     btnSearch.click();
205 }
206
207
208 public void searchByItemID(String itemID) {
```

The screenshot shows the Eclipse IDE interface with the Java EE perspective selected. The top menu bar includes 'File', 'Edit', 'Select', 'Run', 'View', 'Search', 'Help', and 'Quick Access'. The toolbar contains icons for file operations like New, Open, Save, Cut, Copy, Paste, Find, and Delete, along with a 'Run' button. The left sidebar displays the project structure with 'LinksChecker.cl', 'MouseActionUtil', 'SmartRobot.cl', 'URLConverter.cl', 'WebElementWaite', 'Action.java', 'EGClaim.java', 'ClaimType.java', and '*RaiseClaimHome'. The main editor area contains approximately 275 lines of Java code for a 'LinksChecker' class, which handles various search functions (searchByItemId, searchBySellerID, searchByBuyerID) and a 'viewOrderDetails' method. The code uses annotations like @Test and @Before, and imports from packages such as 'org.openqa.selenium.WebElement', 'org.openqa.selenium.support.ui.WebDriverWait', 'org.openqa.selenium.support.ui.ExpectedConditions', and 'java.util.List'. The code also interacts with 'ClaimType.java' and 'EGClaim.java'.

```
209     public void searchByItemId(String itemId) {
210         txtItemId.type(itemId);
211         btnSearch.click();
212     }
213
214
215     public void searchBySellerID(String sellerId) {
216         txtSellerId.type(sellerId);
217         btnSearch.click();
218     }
219
220
221     public void searchByBuyerID(String buyerId) {
222         txtBuyerId.type(buyerId);
223         btnSearch.click();
224     }
225
226
227     public RaiseClaimPage viewOrderDetails(String poisoPayId) throws Exception {
228
229         this.searchByPoisoPayId(String.valueOf(poisoPayId));
230         waitUntilElementIsVisible(driver.findElement(By.xpath("//div[@id='blockUIBlock']")));
231         waitUntilElementIsVisible(driver.findElement(By.xpath("//div[@id='blockUIBlock']")));
232         this.initWebElements();
233         btnSeeDetails(poisoPayId).click();
234         return new RaiseClaimPage();
235     }
236
237
238     public void search(@Role role, EGCSVParams egCSVParams, EbayTransaction es)
239         throws Exception {
240
241         String searchParameter = null;
242
243         if (egCSVParams.getSearchType().equals("Buyer")) {
244             this.searchByItemId((String.valueOf(es.getBuyer())));
245             getSellerName();
246         } else if (egCSVParams.getSearchType().equals("PoisoPay")) {
247             this.searchByPoisoPayId((String.valueOf(es.getPoisoPayId())));
248         } else if (egCSVParams.getSearchType().equals("Item")) {
249             this.searchByItemId(es.getSearchItemList().get(0).getItem().getItemID());
250         } else if (egCSVParams.getSearchType().equals("Seller")) {
251             this.searchBySellerId(es.getSearchItemList().get(0).getSeller().getSellerID());
252             getSellerLoginName();
253         }
254
255         txtSellerId.type(es.getSearchItemList().get(0).getSeller()
256             .getSellerLoginName());
257
258         if (role.getUserRole().equals("CSR2")) {
259             this.searchByPoisoPayId(String.valueOf(es.getPoisoPayId()));
260         } else {
261             btnSearch.click();
262         }
263
264     } else {
265         throw new Exception("Enter a valid search type");
266     }
267     this.searchByPoisoPayId(String.valueOf(es.getPoisoPayId()));
268     waitUntilElementIsVisible(driver.findElement(By.xpath("//div[@id='blockUIBlock']")));
269     waitUntilElementIsVisible(driver.findElement(By.xpath("//div[@id='blockUIBlock']")));
270     this.initWebElements();
271     btnSeeDetails(es.getSearchItemList().get(0).getEgClaim().getClaimId() + "")
272         .click();
273 }
274
275     public void validateLabelLinksPresent(boolean ispresent) {
```

```

239     waitForElementToDisappear(gaxLoader);
240     waitForElementToDisappear(blockUIBlock);
241     this.initWebUXElements();
242     btnSeeDetails(es.getEscrowList().click());
243     return new RaiseClaimPage();
244 }
245
246 public void search(UserRole role, EGCSVParams egparams, EbayTransaction es)
247     throws Exception {
248     String searchParameter = null;
249
250     if (egparams.getSearchType().equals("Buyer")) {
251         this.searchByBuyerID(String.valueOf(es.getBuyer()
252             .getSeller().getUserName())));
253     } else if (es.getBuyerID().equals("PoiSoPay")) {
254         this.searchByPoiSoPayID(String.valueOf(es.getPoiSoPayID()));
255     } else if (egparams.getSearchType().equals("Item")) {
256         this.searchByItemID(es.getEscrowList().get(0).getItemID());
257     } else if (egparams.getSearchType().equals("Seller")) {
258         txtSellerID.type(es.getEscrowList().get(0).getSeller()
259             .getUserName());
260
261         if (role.getUserRole().equals("SR2")) {
262             this.searchByPoiSoPayID(String.valueOf(es.getPoiSoPayID()));
263         } else {
264             btnsSearch.click();
265         }
266     } else {
267         throw new Exception("Enter a valid search type");
268     }
269     Thread.sleep(3000);
270    waitForElementToDisappear(gaxLoader);
271     waitForElementToDisappear(blockUIBlock);
272     this.initWebUXElements();
273     btnSeeDetails(es.getEscrowList().get(0).getEgClaim().getClaimID() + "")
274         .click();
275 }
276
277 public void validateLinksPresent(boolean ispresent) {
278     if (ispresent) {
279         Assertion.assertTrue(isElementPresent(By.xpath("//link[@value='Shipping Label does not present in History']"
280             .getLocator())));
281         Assertion.assertTrue(
282             isElementPresent(By.xpath("//link[@value='Shipping Declaration label does not present in History']"
283                 .getLocator())));
284     } else {
285         Assertion.assertFalse(isElementPresent(By
286             .xpath("//link[@value='Shipping Label present in History']"
287                 .getLocator())));
288         Assertion.assertFalse(isElementPresent(By
289             .xpath("//link[@value='Shipping Declaration label present in History']"
290                 .getLocator())));
291     }
292 }
293
294 }
295
296 }
297
298 }
299 }
300

```

Plan

```

34 package com.ebay.mou.test.ebayeo;
35
36 import java.lang.reflect.Method;
37
38 public class RaiseClaimNegateScenario extends TestPlan {
39     @DataProvider(name = "eGClaimsNotFoundCase", parallel = true)
40     public static Iterator<Object[]> getUserInfo(Method m,
41         ITContext testContext) throws Exception {
42         EasyMockFilter filter = EasyMockFilter.equalsIgnoreCase(
43             m.getName());
44         filter = filter.and(filter, EasyMockFilter.equalsIgnoreCase(
45             TestObject.TEST_JTTF,
46             ContextManager.getCurrentLevelContext(testContext).getSite()));
47
48         LinkedHashMap<String, Class?> classMap = new LinkedHashMap<String, Class?>();
49
50         classMap.put("TestObject", TestObject.class);
51         classMap.put("UserRole", UserRole.class);
52         classMap.put("EbayTransaction", EbayTransaction.class);
53         classMap.put("EscrowDetails", EscrowDetails.class);
54
55         return SpreadSheetUtil.getEntityListFromSpreadSheet(
56             RaiseClaimNegateScenario.class, classMap,
57             "raiseclaimnegateScenarioWSD0.csv", 0, null, filter);
58     }
59
60     @Test(retryAnalyzer = TestRetryAnalyzer.class, groups = {
61         TestType.Regression, Priorities.P2, TestAspect.Positive,
62         "raiseclaimsNotFound"}, dataProvider = "eGClaimsNotFoundCase", description = "Check Claim For Escrow Status")
63     public void raiseClaimSewStatus(String resultMessage,
64         TestObject testObject, UserRole role, EbayTransaction es,
65         EscrowDetails escrowDetails) throws Exception {
66         // List items for raise claim flow
67         PowershipUtil.createTransaction_v1(es);
68         System.out.println("PRID: " + es.getEscrowList().get(0).getPoiSoPayID());
69
70         // checking the escrow date and accordingly update the state for raise
71         Thread.sleep(10000);
72         for (EbayEscrow escrow : es.getEscrowList()) {
73             EClaim eClaim = escrow.getEClaim();
74
75             // update the escrow status
76             EbayIndividModule.updateEscrowStatusEMSEscrowShipmentTable(escrow
77                 .getEscrowID(), escrowDetails.getEscrowStatus()
78                 .getStatusCode());
79
80             ThreadHelper.waitForSeconds(3);
81             if (eClaim != null) {
82                 WebDRIVERUtil.getWebDriver().createWebDriver();
83                 RaiseClaimFailurePage raiseClaimFailurePage = role
84                     .equals(UserRole.CSR) ? eUtil
85                         .raiseClaimNegativeEscrow(null, role, resultMessage) : eUtil.raiseClaimNegative(escrow,
86                         es.getBuyer(), role, resultMessage);
87
88                 RaiseClaimStatusError status = eUtil
89                     .getErrorMessage(resultMessage, es);
90
91                 if (status.getStatus().equalsIgnoreCase(CLAIM_SUCCESS)) {
92                     Assertion.softAssertEqual(
93                         raiseClaimFailurePage.getSuccessMessage(),
94                         status.getMessage(),
95                         "success message and not matching and claim status is= "
96                         + eClaim.getClaimID());
97                 }
98             }
99         }
100     }

```

The screenshot shows the Eclipse IDE interface with the following details:

- Title Bar:** Quick Access, Java EE, Debug.
- Toolbar:** Standard Eclipse toolbar with icons for file operations, search, and preferences.
- Left Sidebar:** Shows the package structure with nodes like ContextManager, Error.java, RaiseClaimSrvRe, RaiseClaimSrvRe, SendOTP.java, Status.java, AbstractPageList, Assertion.class, Context.class, and RaiseClaimNeget.
- Right Sidebar:** Shows the Java EE and Debug perspectives.
- Code Editor:** The main window displays Java code for the `EgClaimHelperPut` class. The code includes imports for `ContextManager`, `LinkedHashMap`, `SpreadSheetUtil`, `TestType`, `Priorities`, `R2`, `TestSpect`, `Positive`, and `EbayUtil`. It contains annotations for `@Test` and `@TestedByAnalyzer`. The code logic involves creating a linked hash map of objects, reading entities from a spreadsheet, and performing assertions on claim status. It also interacts with the eBay module to update escrow status and handle claim creation exceptions.

The screenshot shows the Eclipse IDE interface with the following details:

- Title Bar:** Quick Access, Java EE, Debug.
- Toolbar:** Standard Eclipse toolbar with icons for file operations, search, and preferences.
- Left Margin:** Shows the package structure and file navigation.
- Code Editor:** The main area displays the Java code for `RaiseClaimTestP.java`. The code is annotated with line numbers and includes imports from various eBay and Java utility classes.
- Right Margin:** Shows the Java EE perspective with icons for JBoss Seam, JBoss Seam Test, JBoss Seam Persistence, JBoss Seam Faces, JBoss Seam EJB, JBoss Seam Web, JBoss Seam Resource, JBoss Seam Test Persistence, JBoss Seam Test Faces, JBoss Seam Test EJB, JBoss Seam Test Web, and JBoss Seam Test Resource.

```
1 package com.ebay.moui.test.ebayIn.e6;
2
3 import java.lang.reflect.Method;
4 import java.util.ArrayList;
5 import java.util.List;
6 import java.util.concurrent.TimeUnit;
7 import java.util.List;
8
9 import org.junit.TestContext;
10 import org.junit.annotations.DataProvider;
11 import org.junit.annotations.Test;
12
13 import com.ebay.moui.component.ebayIn.PSServices.util.e6ServiceUtil;
14 import com.ebay.moui.component.ebayIn.common.EbayScrowd;
15 import com.ebay.moui.component.ebayIn.common.EbayTransaction;
16 import com.ebay.moui.component.ebayIn.common.EbayUtil;
17 import com.ebay.moui.component.ebayIn.et.common.EGcsClaim;
18 import com.ebay.moui.component.ebayIn.et.common.EGcsParams;
19 import com.ebay.moui.component.ebayIn.et.common.EscrnNotes;
20 import com.ebay.moui.component.ebayIn.et.common.EscrnStatus;
21 import com.ebay.moui.component.ebayIn.et.common.EscrnUserStatusError;
22 import com.ebay.moui.component.ebayIn.et.common.Status;
23 import com.ebay.moui.component.ebayIn.et.common.UserRole;
24 import com.ebay.moui.component.ebayIn.et.pages.EgcPage;
25 import com.ebay.moui.component.ebayIn.et.pages.GetTransactionDetailsForCSR2;
26 import com.ebay.moui.component.ebayIn.et.pages.InrCreateDispute;
27 import com.ebay.moui.component.ebayIn.et.pages.InrCreateDisputePage;
28 import com.ebay.moui.component.ebayIn.et.pages.RaiseClaimSuccessPage;
29 import com.ebay.moui.component.ebayIn.et.servicePages.Acquisition;
30 import com.ebay.moui.component.ebayIn.et.servicePages.RaiseClaimSvRequest;
31 import com.ebay.moui.component.ebayIn.et.servicePages.RaiseClaimSvResponse;
32 import com.ebay.moui.util.EGBUtil;
33 import com.ebay.moui.util.e6Util;
34 import com.ebay.moui.util.e6Util.EGBUtil;
35 import com.ebay.moui.util.e6Util.EGBUtil;
36 import com.ebay.moui.util.e6Util.EGBUtil;
37 import com.ebay.moui.util.e6Util.RaiseClaimServiceConsumer;
38 import com.ebay.moui.util.ebayInUtils.EbayIndADBModule;
39 import com.ebay.moui.util.ebayInUtils.PowershipUtil;
40 import com.ebay.moui.util.ebayInUtils.SignInUtil;
41 import com.ebay.moui.util.page.cgi3.IbBayGuaranteedebsashboard;
42 import com.ebay.moui.util.page.cgi3.SellerItemDetailPage;
43 import com.ebay.moui.util.page.cgi3.Signin.SigninPage;
44 import com.ebay.moui.util.context.ContextManager;
45 import com.ebay.moui.controller.EasyFilter;
46 import com.ebay.moui.controller.LoginController;
47 import com.ebay.moui.controller.RegisterController;
48 import com.ebay.moui.controller.TestController;
49 import com.ebay.moui.controller.TestRetryAnalyzer;
50 import com.ebay.moui.driver.web.WebDriverDriver;
51 import com.ebay.moui.model.type.ThresholdLevel;
52 import com.ebay.moui.model.type.SpringStatusLevel;
53 import com.ebay.moui.util.internal.entity.TestObject;
54 import com.ebay.moui.util.internal.type.Privileges;
55 import com.ebay.moui.util.internal.type.TestSpec;
56 import com.ebay.moui.util.internal.type.TestType;
57 import com.ebay.powerchip.util.notification.EbayAWBStatusEnum;
58 import com.ibm.icu.util.Calendar;
59
60 public class RaiseClaimTestPlan extends TestPlan {
61
62     // Added by Goutham
63
64     @DataProvider(name = "e6RaiseclaimsNewIssueType", parallel = true)
65     public static Iterator<Object[]> getUserInfoNewIssue(Method m,
66                 ITestContext testContext) throws Exception {
67 }
```

```

07    List<Context> testContext = new ArrayList<Context>();
08    EasyFilter filter = EasyFilter.equalsIgnoreCase(TestObject.TEST_METHOD,
09        "getHome");
10    filter.and(filter, EasyFilter.equalsIgnoreCase(
11        TestObject.TEST_SITE,
12        ContextManager.getTestLevelContext(testContext).getSite()));
13
14    LinkedHashMap<String, Class<?>> classMap = new LinkedHashMap<String, Class<?>>();
15
16    classMap.put("TestObject", TestObject.class);
17    classMap.put("UserRole", UserRole.class);
18    classMap.put("EbayTransaction", EbayTransaction.class);
19    classMap.put("EscrowNotes", EscrowNotes.class);
20
21    return SpreadsheetUtil.getEntityFromSpreadsheet(
22        RaiseClaimTestPlan.class, classMap, "eclaimsNewIssue.csv", 0,
23        null, filter);
24}
25
26 @TestRetryAnalyzer = TestRetryAnalyzer.class, groups = {
27     TestType.Regression.Priorities.PL1, TestAspect.Positive,
28     "RaiseClaim_CSR_Buyer_Flow_New_Issue"},dataProvider = "eGRaiseClaimsNewIssueType", description = "Raise Claim by Buyer and CSRI for New Issue type"
29 public void raiseClaimNewIssue(RegressionDataProvider dataProvider, String testMessage,
30     TestObject testObject, UserRole role, EbayTransaction es,
31     EscrowNotes escrowNotes) throws Exception {
32     Logging.info("STEP-1: list item for raise Claim Flow");
33     PowershipUtil.createTransaction_v1(es);
34     System.out.println("PPID: " + es.getEscrowList().get(0).getPaisoPayID());
35
36     Logging.info("STEP-2: Raise claim by updating the escrow status");
37     Thread.sleep(1000);
38     for (EbayEscrow escrow : es.getEscrowList()) {
39         EGClaim eclaim = escrow.getEgClaim();
40
41         eUtil.updateTransactionStatus(escrowNotes, escrow, es);
42
43         if (eclaim != null) {
44             webDriver.getWebDriver().createWebDriver();
45             RaiseClaimSuccessPage pRaiseClaimSuccessPage = role
46                 .equals(UserRole.CSR) ? eUtil.raiseClaimNewEscrow(
47                     null, role) : eUtil.raiseClaimNewEscrow(
48                     es.getBuyer(), role);
49
50             RaiseClaimStatusError status = eUtil
51                 .getErrorMessage(resultMessage, es);
52
53             if (status.getStatus().equals(
54                 RaiseClaimStatusError.CLAIM_SUCCESS)) {
55                 Integer claimId = 0;
56                 try {
57                     claimId = pRaiseClaimSuccessPage.getClaimID();
58                     eUtil.setClaimID(claimId);
59                 } catch (Exception ex) {
60                     ThreadHelper.waitForSeconds(5);
61                     claimId = Integer.valueOf(EgDBUtil.getClaimID(es
62                         .getEscrowList().get(0).getPaisoPayID()));
63                     eGClaim.setClaimID(claimId);
64                 }
65
66                 if (escrow.getSelectedShippingServiceCode().getBaseC()
67                     .equalsIgnoreCase("Cash On Delivery"));
68                 eGClaim.setPaymentMethod("CASH ON DELIVERY");
69                 else
70                     eGClaim.setPaymentMethod("Credit Card");
71                 eGClaim.setConvenienceFee(EgDBUtil.getCBConvenienceFee(
72                     "CASH ON DELIVERY"));
73             }
74
75             eUtil.setConvenienceFee(EgDBUtil.getCBConvenienceFee(
76                 es.getPaisoPayID()), es.getBuyer());
77             eUtil.getLoginName();
78
79             eUtil.validateClaimID(eGClaim, role, es.getBuyer());
80             try {
81                 String successMessage = pRaiseClaimSuccessPage
82                     .getSuccessMessage();
83                 Assertions.assertEquals(role.getRaiseClaimSuccessPage(),
84                     pRaiseClaimSuccessPage.getSuccessMessage(),
85                     status.getErrorMsg(),
86                     "success message are not matching and claim status is- "
87                     + eGClaim.getClaimID());
88             } catch (Exception ex) {
89                 Assertion.assertTrue(claimId > 0, "claim ID not found");
90                 Log.info("claim id is: " + claimId);
91             } else {
92                 eUtil.validateErrorStatus(status, pRaiseClaimSuccessPage);
93             }
94         }
95     }
96 }
97
98 @DataProvider(name = "eGRaiseClaimOTP", parallel = true)
99 public static Iterator<Object[]> getUserInfoInOTPMethod_m,
100     TestContext testContext throws Exception {
101     EasyFilter filter = EasyFilter.equalsIgnoreCase(TestObject.TEST_METHOD,
102        "getHome");
103     filter.and(filter, EasyFilter.equalsIgnoreCase(
104        TestObject.TEST_SITE,
105        ContextManager.getTestLevelContext(testContext).getSite()));
106
107    LinkedHashMap<String, Class<?>> classMap = new LinkedHashMap<String, Class<?>>();
108
109    classMap.put("TestObject", TestObject.class);
110    classMap.put("UserRole", UserRole.class);
111    classMap.put("EbayTransaction", EbayTransaction.class);
112    classMap.put("EscrowNotes", EscrowNotes.class);
113
114    return SpreadsheetUtil.getEntityFromSpreadsheet(
115        RaiseClaimTestPlan.class, classMap, "eclaimOTP.csv", 0,
116        null, filter);
117}
118
119 @TestRetryAnalyzer = TestRetryAnalyzer.class, groups = {
120     TestType.Regression.Priorities.PL1, TestAspect.Positive,
121     "RaiseClaim_CSR_Buyer_Flow_OTP"},dataProvider = "eGRaiseClaimOTP", description = "Raise Claim by Buyer and CSRI for New Issue type"
122 public void raiseClaimOTPString(String testMessage,
123     TestObject testObject, UserRole role, EbayTransaction es,
124     EscrowNotes escrowNotes) throws Exception {
125     Logging.info("STEP-1: list item for raise Claim Flow");
126     PowershipUtil.createTransaction_v1(es);
127     System.out.println("PPID: " + es.getEscrowList().get(0).getPaisoPayID());
128
129     Logging.info("STEP-2: Raise claim by updating the escrow status");
130     Thread.sleep(1000);
131     for (EbayEscrow escrow : es.getEscrowList()) {
132         EGClaim eclaim = escrow.getEgClaim();
133
134         eUtil.updateTransactionStatus(escrowNotes, escrow, es);
135
136         if (eclaim != null) {
137             webDriver.getWebDriver().createWebDriver();
138             RaiseClaimSuccessPage pRaiseClaimSuccessPage = role
139                 .equals(UserRole.CSR) ? eUtil.raiseClaimNewEscrow(
140                     null, role) : eUtil.raiseClaimNewEscrow(
141                     es.getBuyer(), role);
142
143             RaiseClaimStatusError status = eUtil
144                 .getErrorMessage(resultMessage, es);
145
146             if (status.getStatus().equals(
147                 RaiseClaimStatusError.CLAIM_SUCCESS)) {
148                 Integer claimId = 0;
149                 try {
150                     claimId = pRaiseClaimSuccessPage.getClaimID();
151                     eUtil.setClaimID(claimId);
152                 } catch (Exception ex) {
153                     ThreadHelper.waitForSeconds(5);
154                     claimId = Integer.valueOf(EgDBUtil.getClaimID(es
155                         .getEscrowList().get(0).getPaisoPayID()));
156                     eGClaim.setClaimID(claimId);
157                 }
158
159                 if (escrow.getSelectedShippingServiceCode().getBaseC()
160                     .equalsIgnoreCase("Cash On Delivery"));
161                 eGClaim.setPaymentMethod("CASH ON DELIVERY");
162                 else
163                     eGClaim.setPaymentMethod("Credit Card");
164                 eGClaim.setConvenienceFee(EgDBUtil.getCBConvenienceFee(
165                     "CASH ON DELIVERY"));
166             }
167
168             eUtil.setConvenienceFee(EgDBUtil.getCBConvenienceFee(
169                 es.getPaisoPayID()), es.getBuyer());
170             eUtil.getLoginName();
171
172             eUtil.validateClaimID(eGClaim, role, es.getBuyer());
173             try {
174                 String successMessage = pRaiseClaimSuccessPage
175                     .getSuccessMessage();
176                 Assertions.assertEquals(role.getRaiseClaimSuccessPage(),
177                     pRaiseClaimSuccessPage.getSuccessMessage(),
178                     status.getErrorMsg(),
179                     "success message are not matching and claim status is- "
180                     + eGClaim.getClaimID());
181             } catch (Exception ex) {
182                 Assertion.assertTrue(claimId > 0, "claim ID not found");
183                 Log.info("claim id is: " + claimId);
184             } else {
185                 eUtil.validateErrorStatus(status, pRaiseClaimSuccessPage);
186             }
187         }
188     }
189 }
190
191

```

```

134     eGClaim.setConvenienceFee(EgDBUtil.getCBConvenienceFee(
135         es.getPaisoPayID()), es.getBuyer());
136     eUtil.getLoginName();
137
138     eUtil.validateClaimID(eGClaim, role, es.getBuyer());
139     try {
140         String successMessage = pRaiseClaimSuccessPage
141             .getSuccessMessage();
142         Assertions.assertEquals(role.getRaiseClaimSuccessPage(),
143             pRaiseClaimSuccessPage.getSuccessMessage(),
144             status.getErrorMsg(),
145             "success message are not matching and claim status is- "
146             + eGClaim.getClaimID());
147     } catch (Exception ex) {
148         Assertion.assertTrue(claimId > 0, "claim ID not found");
149         Log.info("claim id is: " + claimId);
150     } else {
151         eUtil.validateErrorStatus(status, pRaiseClaimSuccessPage);
152     }
153 }
154
155 }
156
157 }
158
159 }
160
161
162 @DataProvider(name = "eGRaiseClaimOTP", parallel = true)
163 public static Iterator<Object[]> getUserInfoInOTPMethod_m,
164     TestContext testContext throws Exception {
165     EasyFilter filter = EasyFilter.equalsIgnoreCase(TestObject.TEST_METHOD,
166        "getHome");
167     filter.and(filter, EasyFilter.equalsIgnoreCase(
168        TestObject.TEST_SITE,
169        ContextManager.getTestLevelContext(testContext).getSite()));
170
171    LinkedHashMap<String, Class<?>> classMap = new LinkedHashMap<String, Class<?>>();
172
173    classMap.put("TestObject", TestObject.class);
174    classMap.put("UserRole", UserRole.class);
175    classMap.put("EbayTransaction", EbayTransaction.class);
176    classMap.put("EscrowNotes", EscrowNotes.class);
177
178    return SpreadsheetUtil.getEntityFromSpreadsheet(
179        RaiseClaimTestPlan.class, classMap, "eclaimOTP.csv", 0,
180        null, filter);
181}
182
183 @TestRetryAnalyzer = TestRetryAnalyzer.class, groups = {
184     TestType.Regression.Priorities.PL1, TestAspect.Positive,
185     "RaiseClaim_CSR_Buyer_Flow_OTP"},dataProvider = "eGRaiseClaimOTP", description = "Raise Claim by Buyer and CSRI for New Issue type"
186 public void raiseClaimOTPString(String testMessage,
187     TestObject testObject, UserRole role, EbayTransaction es,
188     EscrowNotes escrowNotes) throws Exception {
189     Logging.info("STEP-1: list item for raise Claim Flow");
190     PowershipUtil.createTransaction_v1(es);
191     System.out.println("PPID: " + es.getEscrowList().get(0).getPaisoPayID());
192
193     Logging.info("STEP-2: Raise claim by updating the escrow status");
194     Thread.sleep(1000);
195     for (EbayEscrow escrow : es.getEscrowList()) {
196         EGClaim eclaim = escrow.getEgClaim();
197
198         eUtil.updateTransactionStatus(escrowNotes, escrow, es);
199
200         if (eclaim != null) {
201             webDriver.getWebDriver().createWebDriver();
202             RaiseClaimSuccessPage pRaiseClaimSuccessPage = role
203                 .equals(UserRole.CSR) ? eUtil.raiseClaimNewEscrow(
204                     null, role) : eUtil.raiseClaimNewEscrow(
205                     es.getBuyer(), role);
206
207             RaiseClaimStatusError status = eUtil
208                 .getErrorMessage(resultMessage, es);
209
210             if (status.getStatus().equals(
211                 RaiseClaimStatusError.CLAIM_SUCCESS)) {
212                 Integer claimId = 0;
213                 try {
214                     claimId = pRaiseClaimSuccessPage.getClaimID();
215                     eUtil.setClaimID(claimId);
216                 } catch (Exception ex) {
217                     ThreadHelper.waitForSeconds(5);
218                     claimId = Integer.valueOf(EgDBUtil.getClaimID(es
219                         .getEscrowList().get(0).getPaisoPayID()));
220                     eGClaim.setClaimID(claimId);
221                 }
222
223                 if (escrow.getSelectedShippingServiceCode().getBaseC()
224                     .equalsIgnoreCase("Cash On Delivery"));
225                 eGClaim.setPaymentMethod("CASH ON DELIVERY");
226                 else
227                     eGClaim.setPaymentMethod("Credit Card");
228                 eGClaim.setConvenienceFee(EgDBUtil.getCBConvenienceFee(
229                     "CASH ON DELIVERY"));
230             }
231
232             eUtil.setConvenienceFee(EgDBUtil.getCBConvenienceFee(
233                 es.getPaisoPayID()), es.getBuyer());
234             eUtil.getLoginName();
235
236             eUtil.validateClaimID(eGClaim, role, es.getBuyer());
237             try {
238                 String successMessage = pRaiseClaimSuccessPage
239                     .getSuccessMessage();
240                 Assertions.assertEquals(role.getRaiseClaimSuccessPage(),
241                     pRaiseClaimSuccessPage.getSuccessMessage(),
242                     status.getErrorMsg(),
243                     "success message are not matching and claim status is- "
244                     + eGClaim.getClaimID());
245             } catch (Exception ex) {
246                 Assertion.assertTrue(claimId > 0, "claim ID not found");
247                 Log.info("claim id is: " + claimId);
248             } else {
249                 eUtil.validateErrorStatus(status, pRaiseClaimSuccessPage);
250             }
251         }
252     }
253 }
254
255
256

```

```
201     if (eGClaim != null) {
202         WebUIUtil.getWebDriver().createWebDriver();
203         RaiseClaimSuccessPage pRaiseClaimSuccessPage = role
204             .equals(UserRole.CSR1) ? eGUtil.raiseClaimOTP(escrow,
205                 es.getBuyer(), role) : eGUtil.raiseClaimOTP(escrow,
206                 es.getBuyer(), role);
207         ...
208         RaiseClaimStatusError status = eGUtil
209             .getErrorMessage(resultMessage, es);
210 
211         if (status.getStatus().equals(
212             RaiseClaimStatusError.CLAIM_SUCCESS)) {
213             Integer claimId = 0;
214             try {
215                 claimId = pRaiseClaimSuccessPage.getClaimId();
216                 eGClaim.setClaimId(claimId);
217             } catch (Exception ex) {
218                 Thread.sleep(10000);
219                 claimId = Integer.valueOf(eGUtil.getClaimID(es
220                     .getEscrowList().get(0).getPaisoPayID()));
221                 eGClaim.setClaimId(claimId);
222             }
223 
224             if (escrow.getSelectedShippingService().getDesc()
225                 .equalsIgnoreCase("Cash On Delivery"))
226                 eGClaim.setPaymentMethod("CASH ON DELIVERY");
227             else
228                 eGClaim.setPaymentMethod("Credit Card");
229 
230             eGClaim.setCODConvenienceFee(CODUtil.getCODConvenienceFee(
231                 es.getPaisoPayID(), es.getBuyer()
232                 .getUserLoginName()));
233             eGUtil.validateClaimID(eGClaim, role, es.getBuyer());
234             try {
235                 String successMessage = pRaiseClaimSuccessPage
236                     .getSuccessMessage();
237                 if (successMessage.equals("Success")) {
238                     Assertions.assertThat(
239                         pRaiseClaimSuccessPage.getSuccessMessage(),
240                         status.getErrorMsg(),
241                         "Success message are not matching and claim status is - "
242                         + eGClaim.getClaimID());
243                 }
244             } catch (Exception ex) {
245                 Assertion.assertThat(claimId > 0, "claim ID not found");
246             }
247             Log.info("claim id is: " + claimId);
248 
249         } else {
250             eGUtil.validateErrorStatus(status, pRaiseClaimSuccessPage);
251         }
252     }
253 
254 }
255 
256 /**
257  * @DataProvider name = "eGRaiseClaim", parallel = true
258  */
259 public static Iterator<Object[]> getuserInfoMethod() {
260     TestContext testContext throws Exception {
261         ...
262     }
263 }
```

```
266     TestContext testContext throws Exception {
267         EasyFilter filter = EasyFilter.ignoreCase(TestObject.TEST_METHOD,
268             TestObject.TEST_PROVIDER);
269         filter = EasyFilter.and(filter, EasyFilter.equalsIgnoreCase(
270             TestObject.TEST_SITE,
271             ContextManager.getContextLevelContext(testContext).getSite()));
272 
273         LinkedHashMap<String, Class<?>> classMap = new LinkedHashMap<String, Class<?>>();
274         classMap.put("TestObject", TestObject.class);
275         classMap.put("EasyFilter", EasyFilter.class);
276         classMap.put("Thread", Thread.class);
277         classMap.put("EbayTransaction", EbayTransaction.class);
278         classMap.put("EscrowNotes", EscrowNotes.class);
279 
280         return SpreadSheetUtil.getEntriesFromSpreadsheet(
281             RaiseClaimTestPlan.class, classMap, "eGclaims.csv", 0, null,
282             Filter);
283     }
284 
285     @TestC(* retryAnalyzer = TestRetryAnalyzer.class, *groups = {
286         TestType.Regression, Priorities.P1, TestAspect.Positive,
287         "RaiseClaim_CSR_Buyer_Flow"},dataProvider = "eGRaiseClaims", description = "Raise Claim by Buyer and CSR")
288     public void verifyRaiseClaimSuccessMessage(TestObject testObject,
289         UserRole role, EbayTransaction es, EscrowNotes escrowNotes)
290     throws Exception {
291         Logging.info("STEP-1: list items for raise Claim flow");
292         PowerMockito.mock(createTransaction.class);
293         System.out
294             .println("PPID= " + es.getEscrowList().get(0).getPaisoPayID());
295 
296         Logging.info("STEP-2: Raise claim by updating the escrow status");
297         Thread.sleep(1000);
298         for (EbayEscrow escrow : es.getEscrowList()) {
299             EGClaim escClaim = escrow.getEGClaim();
300 
301             eGUtil.updateTransactionStatus(escrowNotes, escrow, es);
302 
303             if (escClaim != null) {
304                 WebUIUtil.getWebDriver().createWebDriver();
305                 RaiseClaimSuccessPage pRaiseClaimSuccessPage = role
306                     .equals(UserRole.CSR1) ? eGUtil
307                         .raiseClaimNewNew(
308                             escClaim.getBuyer(), role);
309 
310                 RaiseClaimStatusError status = eGUtil
311                     .getErrorMessage(resultMessage, es);
312 
313                 if (status.getStatus().equals(
314                     RaiseClaimStatusError.CLAIM_SUCCESS)) {
315                     Integer claimId = 0;
316                     try {
317                         claimId = pRaiseClaimSuccessPage.getClaimId();
318                         eGClaim.setClaimId(claimId);
319                     } catch (Exception ex) {
320                         Thread.sleep(10000);
321                         claimId = Integer.valueOf(eGUtil.getClaimID(es
322                             .getEscrowList().get(0).getPaisoPayID()));
323                         eGClaim.setClaimId(claimId);
324 
325                     if (escrow.getSelectedShippingService().getDesc()
326                         .equalsIgnoreCase("Cash On Delivery"))
327                         eGClaim.setPaymentMethod("CASH ON DELIVERY");
328                     else
329                         eGClaim.setPaymentMethod("Credit Card");
330 
331                     eGClaim.setCODConvenienceFee(CODUtil.getCODConvenienceFee(
332                         es.getPaisoPayID(), es.getBuyer()
333                         .getUserLoginName()));
334                 }
335             }
336         }
337     }
338 }
```

```

322         eUtil.validateClaimInfo(es.getClaim(role), es.getBuyer());
323     }
324     try {
325         String successMessage = pRaiseClaimSuccessPage
326             .getSuccessMessage();
327         if (successMessage != null) {
328             Assertion.softAssertEqual(
329                 successMessage,
330                 es.getClaim(role).getSuccessMessage(),
331                 es.getErrorMessage());
332             log.info("success message are not matching and claim status is - "
333                 + es.getClaim(role).getClaimId());
334         }
335     } catch (Exception ex) {
336         Assertion.assertTrue(claimId > 0, "claim ID not found");
337     }
338     log.info("claim id is: " + claimId);
339 }
340 else {
341     eUtil.validateErrorStatus(status, pRaiseClaimSuccessPage);
342 }
343 }
344 }
345 }
346 }
347 }
348 }
349 }
350 }
351 }
352 }
353 }
354 }
355 }
356 }
357 }
358 }
359 }
360 }
361 }
362 }
363 }
364 }
365 }
366 }
367 }
368 }
369 }
370 }
371 }
372 }
373 }
374 }
375 }
376 }
377 }
378 }
379 }
380 }
381 }
382 }
383 }
384 }
385 }
386 }
387 }
388 }
389 }
390 }
391 }
392 }
393 }
394 }
395 }
396 }
397 }
398 }
399 }
400 }
401 }
402 }
403 }
404 }
405 }
406 }
407 }
408 }
409 }
410 }
411 }
412 }
413 }
414 }
415 }
416 }
417 }
418 }
419 }
420 }
421 }
422 }
423 }
424 }
425 }
426 }
427 }
428 }
429 }
430 }
431 }
432 }
433 }
434 }
435 }
436 }
437 }
438 }
439 }
440 }
441 }
442 }
443 }
444 }
445 }
446 }
447 }
448 }
449 }
450 }
451 }
452 }
453 }
454 }
455 }
456 }
457 }
458 }
459 }
460 }
461 }
462 }
463 }
464 }

```

```

398         }
399     }
400     }
401     }
402 }
403 }
404 }
405 }
406 }
407 }
408 }
409 }
410 }
411 }
412 }
413 }
414 }
415 }
416 }
417 }
418 }
419 }
420 }
421 }
422 }
423 }
424 }
425 }
426 }
427 }
428 }
429 }
430 }
431 }
432 }
433 }
434 }
435 }
436 }
437 }
438 }
439 }
440 }
441 }
442 }
443 }
444 }
445 }
446 }
447 }
448 }
449 }
450 }
451 }
452 }
453 }
454 }
455 }
456 }
457 }
458 }
459 }
460 }
461 }
462 }
463 }
464 }

```

```

465     }
466   }
467 }
468 @Test(retryAnalyzer = TestRetryAnalyzer.class, groups = {
469     TestType.Regression, Priorities.P1, TestAspect.Positive,
470     "raiseClaimInResolutioncenter" },dataProvider = "eGRaiseClaims", description = "Raise Claim by CSR1")
471 public void raiseClaimInResolutioncenter(String resultMessage,
472     TestObject testObject, UserRole role, EbayTransaction es,
473     EscrowDetails escrowDetails) throws Exception {
474
475     // List items for raise Claim Flow
476     PowershipUtil.createTransaction_v1(es);
477     Thread.sleep(1000);
478     for (EbayEscrow escrow : es.getEscrowList()) {
479
480         EGClaim eGClaim = escrow.getEgClaim();
481
482         if (eGClaim != null) {
483
484             EboyIndio08Module.updateTransactionDate(escrow.getItem()
485                 .getItemID());
486             String txId = EboyIndio08Module.getTransactionID(escrow
487                 .getItem());
488             String ordLineItem = escrow.getItem().getItemID() + "_"
489                 + txId;
490
491             WebDriver webDriver = new FirefoxDriver();
492
493             InCreatedDispute buyingSummeryPage = new InCreatedDispute(
494                 InCreatedDispute.URL + orderLineItem, true);
495             SignInPage.signInInPages.getBuyer().getUserName();
496             "password");
497
498             buyingSummeryPage.raiseClaim();
499             buyingSummeryPage.close();
500
501             Thread.sleep(10000);
502
503             EGClaim eGClaimDB = new EGClaim(escrow.getItem().getItemID());
504
505             Assertion.assertEquals(eGClaim.getItem().getClaimStatus(),
506                 eGClaimDB.getItemStatus(),
507                 "Claim status does not match");
508             Assertion.assertEquals(eGClaim.getItemType(),
509                 eGClaimDB.getItemType(),
510                 "Item type does not match");
511             Assertion.assertEquals(eGClaim.getItemSellerProductStatus(),
512                 eGClaimDB.getItemSellerProductStatus(),
513                 "Seller product status does not match");
514             Assertion.assertEquals(eGClaim.getItemSellerContactStatus(),
515                 eGClaimDB.getItemSellerContactStatus(),
516                 "Contact status does not match");
517             Assertion.assertEquals(eGClaim.getPreferredResolution(),
518                 eGClaimDB.getPreferredResolution(),
519                 "Preferred resolution does not match");
520
521             Assertion.assertEquals("CREDIT CARD", eGClaimDB
522                 .getPaymentMethod().toUpperCase().trim(),
523                 "payment method does not match");
524
525             if (eGClaim.getConvenienceFee() == null)
526                 Assertion.assertEquals(Double.parseDouble(eGClaimDB
527                     .getConvenienceFee().trim()),
528                     Double.parseDouble("0.00").trim(),
529                     "COD Convenience Fee does not match");
530             else
531                 Assertion.assertEquals(Double.parseDouble(eGClaimDB
532                     .getConvenienceFee().trim()),
533
534             )
535         }
536     }
537 }
538 }
539 }
540
541 // Added by Saranya
542 // An Order is placed for an Item by a buyer. A claim is raised by buyer by
543 // uploading maximum images before Claim Submission.
544
545 @Test(retryAnalyzer = TestRetryAnalyzer.class, groups = {
546     TestType.Regression, Priorities.P2, TestAspect.Positive,
547     "raiseClaimByBuyerByUploadingMaximumImages" },dataProviderClass = CheckClaimTestPlan.class, dataProvider = "eGCheckClaimStatus", description = "RaiseClaimByBuyerByUploadingImages,RaiseClaimByBuyerByUploadin")
548 public void raiseClaimByBuyerByUploadingMaximumImages(
549     TestObject testObject, UserRole role, EbayTransaction es,
550     EGCSParams egcParams) throws Exception {
551
552     PowershipUtil.createTransaction_v1(es);
553
554     Logging.log("Creating claim for the transaction");
555     // User buyer = user.getExistingBuyerForImage();
556
557     for (EbayEscrow escrow : es.getEscrowList()) {
558         EGClaim eGClaim = escrow.getEgClaim();
559         // String buyer = buyer.getBuyer();
560         if (eGClaim != null) {
561
562             EboyIndio08Module.updateTransactionDate(escrow.getItem()
563                 .getItemID());
564             RaiseClaimInSuccessPage raiseClaim = eGUil
565                 .raiseClaimUploadingImage(escrow, es.getBuyer());
566             eGClaim.setClaimID(raiseClaim.getClaimID());
567             raiseClaim.getDriver().quit();
568
569             Thread.sleep(1000);
570         }
571     }
572 }
573 }
574 }
575 }
576
577 // Added by Saranya
578 // An Order is placed for an Item by buyer. A claim is raised by CSR1 User
579 // by uploading maximum images before Claim Submission.
580
581 @Test(retryAnalyzer = TestRetryAnalyzer.class, groups = { "raiseClaimByCSR1ByUploadingMaximumImages" },dataProviderClass = CheckClaimTestPlan.class, dataProvider = "eGCheckClaimStatus", description = "RaiseClaimBy")
582 public void raiseClaimByCSR1ByUploadingMaximumImages(TestObject testObject,
583     UserRole role, EbayTransaction es, EGCSParams egcParams)
584     throws Exception {
585
586     PowershipUtil.createTransaction_v1(es);
587
588     Logging.log("Creating claim for the transaction");
589
590     for (EbayEscrow escrow : es.getEscrowList()) {
591         EGClaim eGClaim = escrow.getEgClaim();
592         if (eGClaim != null) {
593
594             EboyIndio08Module.updateTransactionDate(escrow.getItem()
595                 .getItemID());
596             RaiseClaimInSuccessPage raiseClaim = eGUil
597                 .raiseClaimUploadingImage(escrow, null);
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1888
1889
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2098
2099
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2198
2199
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2289
2290
2291
2292
2293
2294
2295
2296
2297
2297
2298
2299
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2398
2399
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2489
2490
2491
2492
2493
2494
2495
2496
2497
2497
2498
2499
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548

```

The screenshot shows the Eclipse IDE interface with multiple tabs open. The active tab contains Java code for testing claim submission. The code includes annotations for test cases, such as `@Test` and `@TestObject`, and uses assertions from the `Assertion` class. It involves creating a transaction, updating item details, and raising claims with or without images. The code is annotated with descriptions and data providers for each test case.

```
597     .raiseClaimByUploadingImage(escrow, null);
598     eGClaim.setClaimId(craiseClaim.getClaimId());
599     raiseClaim.getDriverO().quit();
600 
601     Thread.sleep(1000);
602 }
603 }
604 }
605 }
606 }
607 }
608 // An Order is placed for an Item by a buyer. A claim is raised by buyer by
609 // uploading maximum images before Claim Submission.
610 @TestRetryAnalyzer = TestRetryAnalyzer.class, groups = { "RaiseClaimByBuyerByUploadingMaximumImagesForNON_PSSeller" },dataProviderClass = CheckClaimTestPlan.class, dataProvider = "eGCheckClaimStatus", description =
611 public void RaiseClaimByBuyerByUploadingMaximumImagesForNON_PSSeller(
612     TestObject testObject, UserRole role, EbayTransaction es,
613     EGWsParams egwspars) throws Exception {
614 
615     PowershipUtil.createTransaction_v1(es);
616 
617     Logging.log("Creating Claim for the transaction");
618     // User buyer = User.getExistingbuyerForImage();
619 
620     for (EbayEscrow escrow : es.getEscrowList()) {
621         Eclaim eclaim = escrow.getEclaim();
622         // String BuyerID= "mgtulunzer";
623         if (eGClaim != null) {
624             EbayIndiaDBModule.updateTransactionDate(escrow.getItem()
625                 .getItemID());
626             RaiseClaimsSuccessPage raiseClaim = eGUtil
627                 .raiseClaimByUploadingImage(escrow, es.getBuyer());
628             eGClaim.setClaimId(craiseClaim.getClaimId());
629             raiseClaim.getDriverO().quit();
630 
631             Thread.sleep(1000);
632         }
633     }
634 }
635 }
636 }
637 }
638 }
639 }
640 }
641 // Added by Smitpal
642 // An Order is placed for an Item by a buyer. A claim is raised by CSR1 User
643 // by uploading maximum images before Claim Submission.
644 @TestRetryAnalyzer = TestRetryAnalyzer.class, groups = { "RaiseClaimByCSR1ByUploadingMaximumImagesForNON_PSSeller" },dataProviderClass = CheckClaimTestPlan.class, dataProvider = "eGCheckClaimStatus", description =
645 public void RaiseClaimByCSR1ByUploadingMaximumImagesForNON_PSSeller(
646     TestObject testObject, UserRole role, EbayTransaction es,
647     EGWsParams egwspars) throws Exception {
648 
649     PowershipUtil.createTransaction_v1(es);
650 
651     Logging.log("Creating claim for the transaction");
652 
653     for (EbayEscrow escrow : es.getEscrowList()) {
654         Eclaim eclaim = escrow.getEclaim();
655         if (eGClaim != null) {
656             EbayIndiaDBModule.updateTransactionDate(escrow.getItem()
657                 .getItemID());
658             RaiseClaimsSuccessPage raiseClaim = eGUtil
659                 .raiseClaimByUploadingImage(escrow, null);
660             eGClaim.setClaimId(craiseClaim.getClaimId());
661             raiseClaim.getDriverO().quit();
662         }
663     }
664 }
```

```
663     raiseClaim.getDriver().quit();
664 }
665     Thread.sleep(1000);
666 }
667 }
668 }
669 }
670 }
671 // Added by @omsphu
672 // An Order is placed for an item by a buyer. A claim is raised by CSR2 User
673 // by uploading maximum images before Claim Submission.
674
675 @Test(retryAnalyzer = TestRetryAnalyzer.class, groups = {"RaiseClaimByCSR2ByUploadingMaximumImagesForNON_PSSeller"}, dataProviderClass = CheckClaimTestPlan.class, dataProvider = "eGCheckClaimStatus", description = "Public user RaiseClaimByCSR2ByUploadingMaximumImagesForNON_PSSeller")
676 public void pRaiseClaimByCSR2ByUploadingMaximumImagesForNON_PSSeller(
677     TestObject testObject, UserRole role, EbayTransaction es,
678     EsCsvParams esCsvParams) throws Exception {
679
680     PowershipUtil.createTransaction_v1();
681
682     Logging.log("Creating claim for the transaction");
683     for (EbayEscrow escrow : es.getEscrowList()) {
684         EgClaim eclaim = escrow.getEgClaim();
685         if (eclaim != null) {
686             RaiseClaimSuccessPage raiseClaim = eUtil.raiseClaim(escrow,
687                     null);
688             eclaim.setClaimId(raiseClaim.getClaimId());
689             raiseClaim.getDriver().quit();
690         }
691     }
692
693     WebDRiver.getWebDRiver().createWebDRiver();
694
695     CSR2HomePage pCSR2HomePage = new CSR2HomePage();
696     pCSR2HomePage.navigateToLoginPage();
697     pCSR2HomePage.clickOnSignIn();
698     pCSR2HomePage.clickOnSignIn();
699     .getEgClaim().getClaimId().toString());
700
701     GetTransactionDetailsForSR2 pGetTransactionDetailsForSR2 = new GetTransactionDetailsForSR2();
702
703     Action action = new Action();
704     action.setAction("click");
705     action.setTarget("pGetTransactionDetailsForSR2.getEgClaim()");
706     pGetTransactionDetailsForSR2.openClaimByCSR2Action(
707         SellerProtectionStatus.findInBuyerCpParams
708         .getSellerProtection());
709
710     pGetTransactionDetailsForSR2.close();
711 }
712
713 @Test(retryAnalyzer = TestRetryAnalyzer.class, groups = {
714     TestType.Regression, Priorities.P1, TestAspect.Positive,
715     {"claimsDashboard"}}, dataProvider = "eGetRaiseClaims", description = "Claims count details in dashboard")
716 public void pClaimsCountInDashboardUsingMessage(
717     TestObject testObject, UserRole role, EbayTransaction es,
718     EscrowDetails escrowDetails) throws Exception {
719
720     SignInPage pSignInPage = new SignInPage();
721     pSignInPage.open();
722     pSignInPage.setUserName("eBaySeller");
723     pSignInPage.setPassword("password");
724
725     EbayGuaranteedDashboard pBuyGuaranteedDashboard = new EbayGuaranteedDashboard(true);
726
727     pBuyGuaranteedDashboard.verifyClaims(es);
728 }
```

```

729     @DataProvider(name = "edRaiseClaimDD", parallel = true)
730     public static Object[][] getuserInfoDDMethod() {
731         Object[][] filter = null;
732         try {
733             EasyFilter filter = EasyFilter.equalIgnoreCase(TestObject.TEST_METHOD,
734                 "getName");
735             filter.addFilter(filter, EasyFilter.equalsIgnoreCase(
736                 TestObject.TEST_ATTRIBUTE,
737                 ContextManager.getContextLevelContext(testContext).getSite()));
738         } catch (Exception e) {
739             log.error("Error while getting context level context");
740         }
741         return filter;
742     }
743     LinkedHashMap<String, Class<?>> classMap = new LinkedHashMap<String, Class<?>>();
744     classMap.put("TestObject", TestObject.class);
745     classMap.put("UserRole", UserRole.class);
746     classMap.put("EbayTransaction", EbayTransaction.class);
747     classMap.put("Escrowoutes", Escrowoutes.class);
748     return SpireSheetUtil.getEntityFromSpreadsheet(
749         RaiseClaimPlan.class, classMap, "eClaimsDD.csv", 0,
750         null, filter);
751 }
752 //Demand Draft validation
753 @Test(criterializer = TestTypeAnalyzer.class, groups = {
754     TestType.Regression, Priorities.P1, TestScript.Positive,
755     "raiseClaim_DD"}, dataProvider = "edRaiseClaimDD", description = "Raise Claim by Buyer For Demand Draft")
756 public void raiseClaimDD(String resultMessage,
757     TestObject testObject, UserRole role, EbayTransaction es,
758     Escrowoutes escrow, Escrowoutes escrowotes) {
759     Logging.info("STEP-1: List Items for raise Claim Flow");
760     PowershipUtil.createTransaction_v1(es);
761     System.out.println("PPID- " + es.getEscrowList().get(0).getPaisaPayID());
762     Logging.info("STEP-2: Raise claim by updating the escrow status");
763     Thread.sleep(1000);
764     for (Escrow escrow : es.getEscrowList()) {
765         EscClaim escClaim = escrow.getEgClaim();
766         egUtil.updateTransactionStatus(escrowotes, escrow, es);
767         if (escClaim != null) {
768             WebODriver.getWebDriver().createWebDriver();
769             RaiseClaimSuccessPage raiseClaimSuccessPage;
770             if (role.equals(UserRole.CSR) ? egUtil.raiseClaimDD(escrow,
771                 null, role) : egUtil.raiseClaimDD(escrow,
772                 es.getBuyer(), role));
773             RaiseClaimStatusError status = egUtil
774                 .getErrorMessage(resultMessage, es);
775             if (status.getStatus().equals(
776                 RaiseClaimStatusError.CLAIM_SUCCESS)) {
777                 Integer claimId = 0;
778                 try {
779                     claimId = raiseClaimSuccessPage.getClaimId();
780                     egClaim.setClaimId(claimId);
781                     if (role.equals(UserRole.BUYER) && egClaim.getRefundPreference().equals(RefundPreferenceEnum.DD)) {
782                         //String benAddress = raiseClaimSuccessPage.getBenAddress();
783                         egClaim.setBeneficiaryAddress(raiseClaimSuccessPage.getBenAddress());
784                     }
785                 } catch (Exception ex) {
786                     Thread.sleep(1000);
787                     claimId = Integer.valueOf(egUtil.getClaimID(es
788                         .getEscrowList().get(0).getPaisaPayID()));
789                     egClaim.setClaimId(claimId);
790                 }
791             }
792             if (escrow.getSelectedShippingService().getDesc()
793                 .equalsIgnoreCase("Cash On Delivery"))
794                 egClaim.setPaymentMethod("CASH ON DELIVERY");
795             else
796                 egClaim.setPaymentMethod("Credit Card");
797             egClaim.setConvenienceFee(GOBUtil.getConvenienceFee(es.getPaisaPayID()), es.getBuyer().getUserLoginName());
798             egUtil.validateClaimID(es.getClaimId(), role, es.getBuyer());
799             if (role.equals(UserRole.BUYER) && egClaim.getRefundPreference().equals(RefundPreferenceEnum.DD)) {
800                 egUtil.validateBenAddressInDB(es);
801             }
802             try {
803                 String successMessage = raiseClaimSuccessPage
804                     .getSuccessMessage();
805                 if (successMessage != null) {
806                     Assertion.softAssertEqual(
807                         raiseClaimSuccessPage.getSuccessMessage(),
808                         successMessage, "Success message and claim status is not matching");
809                 }
810             } catch (Exception ex) {
811                 Assertion.assertTrue(claimId > 0, "claim ID not found");
812             }
813             log.info("claim id is: " + claimId);
814         }
815     }
816     egUtil.validateErrorStatus(status, raiseClaimSuccessPage);
817 }
818 }
819 }
820 }
821 }
822 }
823 }
824 }
825 }
826 }
827 }
828 }
829 }
830 }
831 }
832 }
833 }
834 }
835 }

```

```

769     egUtil.updateTransactionStatus(escrowotes, escrow, es);
770     if (egClaim != null) {
771         WebODriver.getWebDriver().createWebDriver();
772         RaiseClaimSuccessPage raiseClaimSuccessPage;
773         if (role.equals(UserRole.CSR) ? egUtil.raiseClaimDD(escrow,
774             null, role) : egUtil.raiseClaimDD(escrow,
775             es.getBuyer(), role));
776         RaiseClaimStatusError status = egUtil
777             .getErrorMessage(resultMessage, es);
778         if (status.getStatus().equals(
779             RaiseClaimStatusError.CLAIM_SUCCESS)) {
780             Integer claimId = 0;
781             try {
782                 claimId = raiseClaimSuccessPage.getClaimId();
783                 egClaim.setClaimId(claimId);
784                 if (role.equals(UserRole.BUYER) && egClaim.getRefundPreference().equals(RefundPreferenceEnum.DD)) {
785                     //String benAddress = raiseClaimSuccessPage.getBenAddress();
786                     egClaim.setBeneficiaryAddress(raiseClaimSuccessPage.getBenAddress());
787                 }
788             } catch (Exception ex) {
789                 Thread.sleep(1000);
790                 claimId = Integer.valueOf(egUtil.getClaimID(es
791                     .getEscrowList().get(0).getPaisaPayID()));
792                 egClaim.setClaimId(claimId);
793             }
794             if (escrow.getSelectedShippingService().getDesc()
795                 .equalsIgnoreCase("Cash On Delivery"))
796                 egClaim.setPaymentMethod("CASH ON DELIVERY");
797             else
798                 egClaim.setPaymentMethod("Credit Card");
799             egClaim.setConvenienceFee(GOBUtil.getConvenienceFee(es.getPaisaPayID()), es.getBuyer().getUserLoginName());
800             egUtil.validateClaimID(es.getClaimId(), role, es.getBuyer());
801             if (role.equals(UserRole.BUYER) && egClaim.getRefundPreference().equals(RefundPreferenceEnum.DD)) {
802                 egUtil.validateBenAddressInDB(es);
803             }
804             try {
805                 String successMessage = raiseClaimSuccessPage
806                     .getSuccessMessage();
807                 if (successMessage != null) {
808                     Assertion.softAssertEqual(
809                         raiseClaimSuccessPage.getSuccessMessage(),
810                         successMessage, "Success message and claim status is not matching");
811                 }
812             } catch (Exception ex) {
813                 Assertion.assertTrue(claimId > 0, "claim ID not found");
814             }
815             log.info("claim id is: " + claimId);
816         }
817     }
818     egUtil.validateErrorStatus(status, raiseClaimSuccessPage);
819 }
820 }
821 }
822 }
823 }
824 }
825 }
826 }
827 }
828 }
829 }
830 }
831 }
832 }
833 }
834 }
835 }

```

```
1 | #DYNAMIC_SUITE SYSTEM "http://beust.com/testing/testing-1.0.dtd"
2 | <suite name="MultiTestSuite" parallel="methods" verbose="1">
3 |   threads="1" data-provider-thread-count="1">
4 |   <listeners>
5 |     <listener class-name="com.ebay.mau.reporter.HTMLReporter" />
6 |     <listener class-name="com.ebay.mau.util.internal.context.APIContextListener" />
7 |   </listeners>
8 |
9 |   <parameter name="poolType" value="qa" />!-- qa, corp or sandbox -->
10 |   <parameter name="proxy" value="paradise" />
11 |   <parameter name="localProxy" value="true" />!-- true or false -->
12 |   <parameter name="site" value="IN" />
13 |   <parameter name="popHost" value="unicorn.qa.ebay.com" />
14 |
15 |   <parameter name="browser" value="chrome" />!-- firefox, chrome, *explorer -->
16 |   <parameter name="webRunMode" value="locallyInRC" />!-- locallyInRC, ExistingGrid -->
17 |   <parameter name="gridUrl" value="http://grid.stratus.qa.ebay.com:8080/wd/hub" />
18 |   <parameter name="webDriverGrid" value="http://grid.stratus.qa.ebay.com:8080/wd/hub" />
19 |   <parameter name="webSessionGrid" value="12000" />
20 |
21 |   <!-- <parameter name="webDriverGrid" value="http://qa-outodev95:4444/wd/hub" -->
22 |
23 |   <parameter name="openPortInBrowser" value="chrome" />!-- firefox -->
24 |   <parameter name="urlConvertClass" />
25 |   <!-- <parameter name="urlConvert" value="com.ebay.mau.util.internal.web.URLConverter" />
26 |   <!-- <parameter name="urlEndPoint" value="http://signin.qa.ebay.com/admin/v3console/ValidateInternals" -->
27 |
28 |
29 |   <parameter name="skipAudit" value="true" />
30 |   <parameter name="autoCorrelateSessions" value="5" />
31 |   <parameter name="captureSnapshot" value="true" />
32 |   <parameter name="softAssertEnabled" value="true" />
33 |   <parameter name="webSessionTimeout" value="60000" />
34 |
35 |   <parameter name="opisHeadEndPoint" value="http://eazy.qa.ebay.com/ws/api.dll" />
36 |   <parameter name="opisSoapEndPoint" value="http://eazy.qa.ebay.com/wsapi" />
37 |   <parameter name="opisCSlEndPoint" value="http://v3api.core.qa.ebay.com:8080/websvc/eBayCSAPI" />
38 |
39 |   <parameter name="opCertName" value="Admin1app" />
40 |   <parameter name="opAppName" value="Admin1app" />
41 |   <parameter name="opDevName" value="Admin1test1" />
42 |   <parameter name="opDetailLevel" value="4" />
43 |   <parameter name="carServiceURL" value="http://phx5qa01c-302b.stratus.phx.qa.ebay.com:8080/sparksvc/spark/shoppingcart/v1/*" />
44 |
45 |   <parameter name="GEdPoint_kaiseClaim" value="http://www.in.eg.qa.ebay.com/eg/RaiseClaim" />!-- http://www.in.eg.qa.ebay.com/eg/RaiseClaim , https://eg-3.stratus.qa.ebay.com/eg/RaiseClaim -->
46 |
47 | <test name="MultiTest-FF" parallel="methods">
48 |   <groups>
49 |     <run>
50 |       <include names="checkClaimStatus" />!-- ReiseClaim_CSR_Buyer_Flow -->!-- ReiseClaim_DD, ReiseClaim_CSR_Buyer_Flow_New_Issue, ReiseClaim_Refund, checkClaimStatus, applySellerProtection -->
51 |       <!--<include name="buying00" /> <include name="BuyingMultipleBidItem" /> -->
52 |     </run>
53 |
54 |     <!-- <exclude name="BuyingCFFlow" /> <exclude name="validateAddress" /> <exclude name="MSKU_Checkout_Flows" /> -->
55 |     <!--<exclude name="validateEmailAddress" /> <exclude name="MSKU_Checkout_Flows" /> -->
56 |   </groups>
57 |   <packages>
58 |     <package name="com.ebay.mau.test.ebayIn.eg" /> <!-- com.ebay.mau.test.ebayIn.eg, -->
59 |
60 |   </packages>
61 |   </test>
62 |   <!-- <test name="Multi - lit" parallel="methods"> -->
63 |   <!-- <parameter name="browser" value="chrome" /> -->
64 | 
```

```
4<!-->
5<!-->
6<!-->
7<!-->
8<!-->
9<!-->
10<!-->
11<!-->
12<!-->
13<!-->
14<!-->
15<!-->
16<!-->
17<!-->
18<!-->
19<!-->
20<!-->
21<!-->
22<!-->
23<!-->
24<!-->
25<!-->
26<!-->
27<!-->
28<!-->
29<!-->
30<!-->
31<!-->
32<!-->
33<!-->
34<!-->
35<!-->
36<!-->
37<!-->
38<!-->
39<!-->
40<!-->
41<!-->
42<!-->
43<!-->
44<!-->
45<!-->
46<!-->
47<!-->
48<!-->
49<!-->
50<!-->
51<!-->
52<!-->
53<!-->
54<!-->
55<!-->
56<!-->
57<!-->
58<!-->
59<!-->
60<!-->
61<!-->
62<!-->
63<!-->
64<!-->
65<!-->
66<!-->
67<!-->
68<!-->
69<!-->
70<!-->
71<!-->
72<!-->
73<!-->
74<!-->
75<!-->
76<!-->
77<!-->
78<!-->
79<!-->
80<!-->
81<!-->
82<!-->
83<!-->
84<!-->
85<!-->
86<!-->
87<!-->
88<!-->
89<!-->
90<!-->
91<!-->
```

eClaimsNewIssue											
Home	Insert	Page Layout	Formulas	Data	Review	View	Wrap Text	General	Conditional Formatting	Format as Table	Cell Styles
C14	B	raiseClaim-BUYER-INAD-DIFF_COLOR-REPLACEMENT-SUCCESS-013									
1	TestObject.TestCaseId	TestObject.TestMethod	TestObject.TestTitle	C	D	E	F	G	H	I	J
2	INR-001	raiseClaim-NewIssueType	raiseClaim-BUYER-INR-ITEMBACK-INR.SUCCESS-001	IN	BUYER	\$50017	AMEX.CREDIT.CARD			FALSE	
3	INR-002	raiseClaim-NewIssueType	raiseClaim-BUYER-INR-MONEYBACK-INR.SUCCESS-002	IN	CSR1	\$50017	AMEX.CREDIT.CARD			FALSE	
4	INR-003	raiseClaim-NewIssueType	raiseClaim-CSRL-INR-ITEMBACK-INR.SUCCESS-003	IN	CSR1	\$50017	AMEX.CREDIT.CARD			FALSE	
5	INR-004	raiseClaim-NewIssueType	raiseClaim-CSRL-INR-MONEYBACK-INR.SUCCESS-004	IN	CSR1	\$50017	AMEX.CREDIT.CARD			FALSE	
6	INR-005	raiseClaim-NewIssueType	raiseClaim-BUYER-INR-ITEMBACK-INR._BEFORE7_FAILURE-005	IN	BUYER	\$50017	AMEX.CREDIT.CARD			FALSE	
7	INR-006	raiseClaim-NewIssueType	raiseClaim-BUYER-INR-MONEYBACK-INR._BEFORE7_FAILURE-006	IN	BUYER	\$50017	AMEX.CREDIT.CARD			FALSE	
8	INR-007	raiseClaim-NewIssueType	raiseClaim-CSRL-INR-ITEMBACK-INR._BEFORE7_FAILURE-007	IN	CSR1	\$50017	AMEX.CREDIT.CARD			FALSE	
9	INR-008	raiseClaim-NewIssueType	raiseClaim-CSRL-INR-MONEYBACK-INR._BEFORE7_FAILURE-008	IN	CSR1	\$50017	AMEX.CREDIT.CARD			FALSE	
10	INR-009	raiseClaim-NewIssueType	raiseClaim-BUYER-INR-ITEMBACK-INR._AFTER3_FAILURE-009	IN	BUYER	\$50017	AMEX.CREDIT.CARD			FALSE	
11	INR-010	raiseClaim-NewIssueType	raiseClaim-BUYER-INR-MONEYBACK-INR._AFTER3_FAILURE-010	IN	BUYER	\$50017	AMEX.CREDIT.CARD			FALSE	
12	INR-011	raiseClaim-NewIssueType	raiseClaim-CSRL-INR-ITEMBACK-INR.SUCCESS-011	IN	CSR1	\$50017	AMEX.CREDIT.CARD			FALSE	
13	INR-012	raiseClaim-NewIssueType	raiseClaim-CSRL-INR-MONEYBACK-INR.SUCCESS-012	IN	CSR1	\$50017	AMEX.CREDIT.CARD			FALSE	
14	CLAM-001	raiseClaim-NewIssueType	raiseClaim-BUYER-INAD-DIFF_COLOR-REPLACEMENT-SUCCESS-013	IN	BUYER	\$50017	AMEX.CREDIT.CARD			FALSE	
15	CLAM-002	raiseClaim-NewIssueType	raiseClaim-BUYER-INAD-DIFF_MODEL_MONEYBACK-SUCCESS-014	IN	BUYER	\$50017	AMEX.CREDIT.CARD			FALSE	
16	CLAM-003	raiseClaim-NewIssueType	raiseClaim-BUYER-INAD-DIFF_MODEL_MONEYBACK-SUCCESS-015	IN	BUYER	\$50017	AMEX.CREDIT.CARD			FALSE	
17	CLAM-004	raiseClaim-NewIssueType	raiseClaim-BUYER-INAD-DIFF_MODEL_MODEL-MONEYBACK-SUCCESS-016	IN	BUYER	\$50017	AMEX.CREDIT.CARD			FALSE	
18	CLAM-005	raiseClaim-NewIssueType	raiseClaim-BUYER-INAD-DIFF_MODEL_MODEL-MONEYBACK-SUCCESS-017	IN	BUYER	\$50017	AMEX.CREDIT.CARD			FALSE	
19	CLAM-006	raiseClaim-NewIssueType	raiseClaim-BUYER-INAD_EXPIRED-MONEYBACK-SUCCESS-018	IN	BUYER	\$50017	AMEX.CREDIT.CARD			FALSE	
20	CLAM-007	raiseClaim-NewIssueType	raiseClaim-BUYER-INAD_EXPIRED-REPLACEMENT-SUCCESS-019	IN	BUYER	\$50017	AMEX.CREDIT.CARD			FALSE	
21	CLAM-008	raiseClaim-NewIssueType	raiseClaim-BUYER-INAD_DIFF_PROD-MONEYBACK-SUCCESS-020	IN	BUYER	\$50017	AMEX.CREDIT.CARD			FALSE	
22	CLAM-009	raiseClaim-NewIssueType	raiseClaim-BUYER-INAD_ODOUR-REPLACEMENT-SUCCESS-021	IN	BUYER	\$50017	AMEX.CREDIT.CARD			FALSE	
23	CLAM-010	raiseClaim-NewIssueType	raiseClaim-BUYER-INAD_ODOUR-MONEYBACK-SUCCESS-022	IN	BUYER	\$50017	AMEX.CREDIT.CARD			FALSE	
24	CLAM-011	raiseClaim-NewIssueType	raiseClaim-BUYER-INAD_QUALITY-REPLACEMENT-SUCCESS-023	IN	BUYER	\$50017	AMEX.CREDIT.CARD			FALSE	
25	CLAM-012	raiseClaim-NewIssueType	raiseClaim-BUYER-INAD_QUALITY-MONEYBACK-SUCCESS-024	IN	BUYER	\$50017	AMEX.CREDIT.CARD			FALSE	
26	CLAM-013	raiseClaim-NewIssueType	raiseClaim-BUYER-INAD_SEAL-REPLACEMENT-SUCCESS-025	IN	BUYER	\$50017	AMEX.CREDIT.CARD			FALSE	
27	CLAM-014	raiseClaim-NewIssueType	raiseClaim-BUYER-INAD_SEAL-MONEYBACK-SUCCESS-026	IN	BUYER	\$50017	AMEX.CREDIT.CARD			FALSE	
28	CLAM-015	raiseClaim-NewIssueType	raiseClaim-BUYER-INAD_USED-REPLACEMENT-SUCCESS-027	IN	BUYER	\$50017	AMEX.CREDIT.CARD			FALSE	
29	CLAM-016	raiseClaim-NewIssueType	raiseClaim-BUYER-INAD_USED-MONEYBACK-SUCCESS-028	IN	BUYER	\$50017	AMEX.CREDIT.CARD			FALSE	
30	CLAM-017	raiseClaim-NewIssueType	raiseClaim-BUYER-INAD_FADE-REPLACEMENT-SUCCESS-029	IN	BUYER	\$50017	AMEX.CREDIT.CARD			FALSE	
31	CLAM-018	raiseClaim-NewIssueType	raiseClaim-BUYER-INAD_FADE_MODEL-MONEYBACK-SUCCESS-030	IN	BUYER	\$50017	AMEX.CREDIT.CARD			FALSE	
32	CLAM-019	raiseClaim-NewIssueType	raiseClaim-BUYER-INAD_STAINED-MONEYBACK-SUCCESS-031	IN	BUYER	\$50017	AMEX.CREDIT.CARD			FALSE	
33	CLAM-020	raiseClaim-NewIssueType	raiseClaim-BUYER-INAD_STAINED-MONEYBACK-SUCCESS-032	IN	BUYER	\$50017	AMEX.CREDIT.CARD			FALSE	
34	CLAM-021	raiseClaim-NewIssueType	raiseClaim-CSRL-INAD_DIFF_COLOR-REPLACEMENT-SUCCESS-033	IN	CSR1	\$50017	AMEX.CREDIT.CARD			FALSE	
35	CLAM-022	raiseClaim-NewIssueType	raiseClaim-CSRL-INAD_DIFF_COLOR-MONEYBACK-SUCCESS-034	IN	CSR1	\$50017	AMEX.CREDIT.CARD			FALSE	
36	CLAM-023	raiseClaim-NewIssueType	raiseClaim-CSRL-INAD_DIFF_MODEL-REPLACEMENT-SUCCESS-035	IN	CSR1	\$50017	AMEX.CREDIT.CARD			FALSE	
37	CLAM-024	raiseClaim-NewIssueType	raiseClaim-CSRL-INAD_DIFF_MODEL-MONEYBACK-SUCCESS-036	IN	CSR1	\$50017	AMEX.CREDIT.CARD			FALSE	
38	CLAM-025	raiseClaim-NewIssueType	raiseClaim-CSRL-INAD_DIFF_MODEL-MONEYBACK-SUCCESS-037	IN	CSR1	\$50017	AMEX.CREDIT.CARD			FALSE	
39	CLAM-026	raiseClaim-NewIssueType	raiseClaim-CSRL-INAD_EXPIRED-MONEYBACK-SUCCESS-038	IN	CSR1	\$50017	AMEX.CREDIT.CARD			FALSE	
40	CLAM-027	raiseClaim-NewIssueType	raiseClaim-CSRL-INAD_EXPIRED-REPLACEMENT-SUCCESS-039	IN	CSR1	\$50017	AMEX.CREDIT.CARD			FALSE	
41	CLAM-028	raiseClaim-NewIssueType	raiseClaim-CSRL-INAD_DIFF_PROD-MONEYBACK-SUCCESS-040	IN	CSR1	\$50017	AMEX.CREDIT.CARD			FALSE	
42	CLAM-029	raiseClaim-NewIssueType	raiseClaim-CSRL-INAD_ODOUR-REPLACEMENT-SUCCESS-041	IN	CSR1	\$50017	AMEX.CREDIT.CARD			FALSE	
43	CLAM-030	raiseClaim-NewIssueType	raiseClaim-CSRL-INAD_ODOUR-MONEYBACK-SUCCESS-042	IN	CSR1	\$50017	AMEX.CREDIT.CARD			FALSE	
44	CLAM-031	raiseClaim-NewIssueType	raiseClaim-CSRL-INAD_QUALITY-REPLACEMENT-SUCCESS-043	IN	CSR1	\$50017	AMEX.CREDIT.CARD			FALSE	
45	CLAM-032	raiseClaim-NewIssueType	raiseClaim-CSRL-INAD_QUALITY-MONEYBACK-SUCCESS-044	IN	CSR1	\$50017	AMEX.CREDIT.CARD			FALSE	

eClaimsNewIssue												
raiseClaim-BUYER-INAD-DIFF_COLOR-REPLACEMENT-SUCCESS-013												
	I	J	K	L	M	N	O					
1	w	EbayTransaction.Buyer.dayPhone1	EbayTransaction.Buyer.nightPhone1	EbayTransaction.escrowList.0.Seller.UserLoginName	EbayTransaction.escrowList.0.SelectedShippingService	EbayTransaction.escrowList.0.ItemTemplate	EbayTransaction.escrowList.0.itemPrice	EbayTransaction.escrowList.0.C				
2		9999999999	8888888888	ps_automation_001	FLAT	PS_INBIN	1500	NA				
3		9999999999	8888888888	ps_automation_001	FLAT	PS_INBIN	1500	NA				
4		9999999999	8888888888	ps_automation_001	FLAT	PS_INBIN	1500	NA				
5		9999999999	8888888888	ps_automation_001	FLAT	PS_INBIN	1500	NA				
6		9999999999	8888888888	ps_automation_001	FLAT	PS_INBIN	1500	NA				
7		9999999999	8888888888	ps_automation_001	FLAT	PS_INBIN	1500	NA				
8		9999999999	8888888888	ps_automation_001	FLAT	PS_INBIN	1500	NA				
9		9999999999	8888888888	ps_automation_001	FLAT	PS_INBIN	1500	NA				
10		9999999999	8888888888	ps_automation_001	FLAT	PS_INBIN	1500	NA				
11		9999999999	8888888888	ps_automation_001	FLAT	PS_INBIN	1500	NA				
12		9999999999	8888888888	ps_automation_001	FLAT	PS_INBIN	1500	NA				
13		9999999999	8888888888	ps_automation_001	FLAT	PS_INBIN	1500	NA				
14		9999999999	8888888888	ps_automation_001	FLAT	PS_INBIN	1500	NA				
15		9999999999	8888888888	ps_automation_001	FLAT	PS_INBIN	1500	NA				
16		9999999999	8888888888	ps_automation_001	FLAT	PS_INBIN	1500	NA				
17		9999999999	8888888888	ps_automation_001	FLAT	PS_INBIN	1500	NA				
18		9999999999	8888888888	ps_automation_001	FLAT	PS_INBIN	1500	NA				
19		9999999999	8888888888	ps_automation_001	FLAT	PS_INBIN	1500	NA				
20		9999999999	8888888888	ps_automation_001	FLAT	PS_INBIN	1500	NA				
21		9999999999	8888888888	ps_automation_001	FLAT	PS_INBIN	1500	NA				
22		9999999999	8888888888	ps_automation_001	FLAT	PS_INBIN	1500	NA				
23		9999999999	8888888888	ps_automation_001	FLAT	PS_INBIN	1500	NA				
24		9999999999	8888888888	ps_automation_001	FLAT	PS_INBIN	1500	NA				
25		9999999999	8888888888	ps_automation_001	FLAT	PS_INBIN	1500	NA				
26		9999999999	8888888888	ps_automation_001	FLAT	PS_INBIN	1500	NA				
27		9999999999	8888888888	ps_automation_001	FLAT	PS_INBIN	1500	NA				
28		9999999999	8888888888	ps_automation_001	FLAT	PS_INBIN	1500	NA				
29		9999999999	8888888888	ps_automation_001	FLAT	PS_INBIN	1500	NA				
30		9999999999	8888888888	ps_automation_001	FLAT	PS_INBIN	1500	NA				
31		9999999999	8888888888	ps_automation_001	FLAT	PS_INBIN	1500	NA				
32		9999999999	8888888888	ps_automation_001	FLAT	PS_INBIN	1500	NA				
33		9999999999	8888888888	ps_automation_001	FLAT	PS_INBIN	1500	NA				
34		9999999999	8888888888	ps_automation_001	FLAT	PS_INBIN	1500	NA				
35		9999999999	8888888888	ps_automation_001	FLAT	PS_INBIN	1500	NA				
36		9999999999	8888888888	ps_automation_001	FLAT	PS_INBIN	1500	NA				
37		9999999999	8888888888	ps_automation_001	FLAT	PS_INBIN	1500	NA				
38		9999999999	8888888888	ps_automation_001	FLAT	PS_INBIN	1500	NA				
39		9999999999	8888888888	ps_automation_001	FLAT	PS_INBIN	1500	NA				
40		9999999999	8888888888	ps_automation_001	FLAT	PS_INBIN	1500	NA				
41		9999999999	8888888888	ps_automation_001	FLAT	PS_INBIN	1500	NA				
42		9999999999	8888888888	ps_automation_001	FLAT	PS_INBIN	1500	NA				
43		9999999999	8888888888	ps_automation_001	FLAT	PS_INBIN	1500	NA				
44		9999999999	8888888888	ps_automation_001	FLAT	PS_INBIN	1500	NA				
45		oooooooooooo	oooooooooooo	ps_automation_001	FLAT	PS_INBIN	1500	NA				

O	P	Q	R	EbayTransaction.escrowList.0.CodStatus	EbayTransaction.escrowList.0.EgClaim.categoryId	EbayTransaction.escrowList.0.EgClaim.claimType	EbayTransaction.escrowList.0.EgClaim.subIssue	EbayTransaction.escrowList.0.EgClaim.subIssueList
1	ce	EbayTransaction.escrowList.0.CodStatus	ITEM_NOT_RECEIVED	2	500	NA	ITEM_NOT_RECEIVED	ITEM_NOT_RECEIVED
2	OPEN		ITEM_NOT_RECEIVED	3	500	NA	ITEM_NOT_RECEIVED	
4	OPEN		ITEM_NOT_RECEIVED	4	500	NA	ITEM_NOT_RECEIVED	
5	OPEN		ITEM_NOT_RECEIVED	5	500	NA	ITEM_NOT_RECEIVED	
6	OPEN		ITEM_NOT_RECEIVED	6	500	NA	ITEM_NOT_RECEIVED	
7	OPEN		ITEM_NOT_RECEIVED	7	500	NA	ITEM_NOT_RECEIVED	
8	OPEN		ITEM_NOT_RECEIVED	8	500	NA	ITEM_NOT_RECEIVED	
9	OPEN		ITEM_NOT_RECEIVED	9	500	NA	ITEM_NOT_RECEIVED	
10	OPEN		ITEM_NOT_RECEIVED	10	500	NA	ITEM_NOT_RECEIVED	
11	OPEN		ITEM_NOT_RECEIVED	11	500	NA	ITEM_NOT_RECEIVED	
12	OPEN		ITEM_NOT_RECEIVED	12	500	NA	ITEM_NOT_RECEIVED	
13	OPEN		ITEM_NOT_RECEIVED	13	500	NA	ITEM_NOT_RECEIVED	
14	OPEN		ITEM_NOT_AS_DESCRIBED	14	500	NA	DIFFERENT_COLOUR_SIZE_PATTERN_	DIFFERENT_COLOUR_SIZE_PATTERN_,DIFFERENT_MODEL_CONFIGURATION;EXPIRED_PRODUCT_RECEIVED_INAD;
15	OPEN		ITEM_NOT_AS_DESCRIBED	15	500	NA	DIFFERENT_COLOUR_SIZE_PATTERN_	
16	OPEN		ITEM_NOT_AS_DESCRIBED	16	500	NA	DIFFERENT_MODEL_CONFIGURATION	
17	OPEN		ITEM_NOT_AS_DESCRIBED	17	500	NA	DIFFERENT_MODEL_CONFIGURATION	
18	OPEN		ITEM_NOT_AS_DESCRIBED	18	500	NA	EXPIRED_PRODUCT_RECEIVED_INAD	
19	OPEN		ITEM_NOT_AS_DESCRIBED	19	500	NA	EXPIRED_PRODUCT_RECEIVED_INAD	
20	OPEN		ITEM_NOT_AS_DESCRIBED	20	500	NA	DIFFERENT_PRODUCT	
21	OPEN		ITEM_NOT_AS_DESCRIBED	21	500	NA	DIFFERENT_PRODUCT	
22	OPEN		ITEM_NOT_AS_DESCRIBED	22	500	NA	HAS_AN_ODOUR	
23	OPEN		ITEM_NOT_AS_DESCRIBED	23	500	NA	HAS_AN_ODOUR	
24	OPEN		ITEM_NOT_AS_DESCRIBED	24	500	NA	POOR_QUALITY	
25	OPEN		ITEM_NOT_AS_DESCRIBED	25	500	NA	POOR_QUALITY	
26	OPEN		ITEM_NOT_AS_DESCRIBED	26	500	NA	SEAL_BROKEN	
27	OPEN		ITEM_NOT_AS_DESCRIBED	27	500	NA	SEAL_BROKEN	
28	OPEN		ITEM_NOT_AS_DESCRIBED	28	500	NA	USED_PRODUCT	
29	OPEN		ITEM_NOT_AS_DESCRIBED	29	500	NA	USED_PRODUCT	
30	OPEN		ITEM_NOT_AS_DESCRIBED	30	500	NA	FAKE_COUNTERFEIT	
31	OPEN		ITEM_NOT_AS_DESCRIBED	31	500	NA	FAKE_COUNTERFEIT	
32	OPEN		ITEM_NOT_AS_DESCRIBED	32	500	NA	STAINED_INAD	
33	OPEN		ITEM_NOT_AS_DESCRIBED	33	500	NA	STAINED_INAD	
34	OPEN		ITEM_NOT_AS_DESCRIBED	34	500	NA	STAINED_INAD	
35	OPEN		ITEM_NOT_AS_DESCRIBED	35	500	NA	DIFFERENT_COLOUR_SIZE_PATTERN_	DIFFERENT_COLOUR_SIZE_PATTERN_,DIFFERENT_MODEL_CONFIGURATION;EXPIRED_PRODUCT_RECEIVED_INAD;
36	OPEN		ITEM_NOT_AS_DESCRIBED	36	500	NA	DIFFERENT_COLOUR_SIZE_PATTERN_	
37	OPEN		ITEM_NOT_AS_DESCRIBED	37	500	NA	DIFFERENT_MODEL_CONFIGURATION	
38	OPEN		ITEM_NOT_AS_DESCRIBED	38	500	NA	DIFFERENT_MODEL_CONFIGURATION	
39	OPEN		ITEM_NOT_AS_DESCRIBED	39	500	NA	EXPIRED_PRODUCT_RECEIVED_INAD	
40	OPEN		ITEM_NOT_AS_DESCRIBED	40	500	NA	EXPIRED_PRODUCT_RECEIVED_INAD	
41	OPEN		ITEM_NOT_AS_DESCRIBED	41	500	NA	DIFFERENT_PRODUCT	
42	OPEN		ITEM_NOT_AS_DESCRIBED	42	500	NA	DIFFERENT_PRODUCT	
43	OPEN		ITEM_NOT_AS_DESCRIBED	43	500	NA	HAS_AN_ODOUR	
44	OPEN		ITEM_NOT_AS_DESCRIBED	44	500	NA	POOR_QUALITY	
45	OPEN		ITEM_NOT_AS_DESCRIBED	45	500	NA	POOR_QUALITY	

T	U	V	W	X	eGclaimsNewIssue
1	EbayTransaction.escrowList.0.EgClaim.claimStatus	EbayTransaction.escrowList.0.EgClaim.isInvoiceReceived	EbayTransaction.escrowList.0.EgClaim.isImageMandatory	EbayTransaction.escrowList.0.EgClaim.preferredResolution	EbayTransaction.escrowList.0.EgClaim.refundPreference
2	OPEN	NA	NA	I_WANT_THE_ITEM	ORIGINAL_MODE_OF_PAYMENT
3	OPEN	NA	NA	I_WANT_MY_MONEY_BACK	NOT_TRIED_1
4	OPEN	NA	NA	I_WANT_THE_ITEM	NOT_TRIED_1
5	OPEN	NA	NA	I_WANT_MY_MONEY_BACK	ORIGINAL_MODE_OF_PAYMENT
6	OPEN	NA	NA	I_WANT_A_REPLACEMENT	NOT_TRIED_1
7	OPEN	NA	NA	I_WANT_MY_MONEY_BACK	NOT_TRIED_1
8	OPEN	NA	NA	I_WANT_A_REPLACEMENT	NOT_TRIED_1
9	OPEN	NA	NA	I_WANT_MY_MONEY_BACK	NOT_TRIED_1
10	OPEN	NA	NA	I_WANT_A_REPLACEMENT	NOT_TRIED_1
11	OPEN	NA	NA	I_WANT_MY_MONEY_BACK	NOT_TRIED_1
12	OPEN	NA	NA	I_WANT_A_REPLACEMENT	NOT_TRIED_1
13	OPEN	NA	NA	I_WANT_MY_MONEY_BACK	NOT_TRIED_1
14	OPEN	NA	YES	I_WANT_A_REPLACEMENT	NOT_TRIED_1
15	OPEN	NA	YES	I_WANT_MY_MONEY_BACK	ORIGINAL_MODE_OF_PAYMENT
16	OPEN	NA	YES	I_WANT_A_REPLACEMENT	NOT_TRIED_1
17	OPEN	NA	YES	I_WANT_MY_MONEY_BACK	ORIGINAL_MODE_OF_PAYMENT
18	OPEN	NA	YES	I_WANT_A_REPLACEMENT	NOT_TRIED_1
19	OPEN	NA	YES	I_WANT_MY_MONEY_BACK	ORIGINAL_MODE_OF_PAYMENT
20	OPEN	NA	YES	I_WANT_A_REPLACEMENT	NOT_TRIED_1
21	OPEN	NA	YES	I_WANT_MY_MONEY_BACK	ORIGINAL_MODE_OF_PAYMENT
22	OPEN	NA	NO	I_WANT_A_REPLACEMENT	NOT_TRIED_1
23	OPEN	NA	NO	I_WANT_MY_MONEY_BACK	ORIGINAL_MODE_OF_PAYMENT
24	OPEN	NA	NO	I_WANT_A_REPLACEMENT	NOT_TRIED_1
25	OPEN	NA	NO	I_WANT_MY_MONEY_BACK	ORIGINAL_MODE_OF_PAYMENT
26	OPEN	NA	YES	I_WANT_A_REPLACEMENT	NOT_TRIED_1
27	OPEN	NA	YES	I_WANT_A_REPLACEMENT	ORIGINAL_MODE_OF_PAYMENT
28	OPEN	NA	YES	I_WANT_A_REPLACEMENT	NOT_TRIED_1
29	OPEN	NA	YES	I_WANT_MY_MONEY_BACK	ORIGINAL_MODE_OF_PAYMENT
30	OPEN	NA	YES	I_WANT_A_REPLACEMENT	NOT_TRIED_1
31	OPEN	NA	YES	I_WANT_A_REPLACEMENT	ORIGINAL_MODE_OF_PAYMENT
32	OPEN	NA	YES	I_WANT_A_REPLACEMENT	NOT_TRIED_1
33	OPEN	NA	YES	I_WANT_A_REPLACEMENT	ORIGINAL_MODE_OF_PAYMENT
34	OPEN	NA	NA	I_WANT_A_REPLACEMENT	NOT_TRIED_1
35	OPEN	NA	NA	I_WANT_MY_MONEY_BACK	ORIGINAL_MODE_OF_PAYMENT
36	OPEN	NA	NA	I_WANT_A_REPLACEMENT	NOT_TRIED_1
37	OPEN	NA	NA	I_WANT_MY_MONEY_BACK	ORIGINAL_MODE_OF_PAYMENT
38	OPEN	NA	NA	I_WANT_A_REPLACEMENT	NOT_TRIED_1
39	OPEN	NA	NA	I_WANT_MY_MONEY_BACK	ORIGINAL_MODE_OF_PAYMENT
40	OPEN	NA	NA	I_WANT_A_REPLACEMENT	NOT_TRIED_1
41	OPEN	NA	NA	I_WANT_MY_MONEY_BACK	ORIGINAL_MODE_OF_PAYMENT
42	OPEN	NA	NA	I_WANT_A_REPLACEMENT	NOT_TRIED_1
43	OPEN	NA	NA	I_WANT_MY_MONEY_BACK	ORIGINAL_MODE_OF_PAYMENT
44	OPEN	NA	NA	I_WANT_A_REPLACEMENT	NOT_TRIED_1
45	OPEN	NA	NA	I_WANT_MY_MONEY_BACK	ORIGINAL_MODE_OF_PAYMENT

	Z	AA	AB	AC
1	EbayTransaction.escrowList.0.EgClaim.sellerContactStatus	EbayTransaction.escrowList.0.EgClaim.BuyerRemarks	EbayTransaction.escrowList.1.SelectedShippingService	EbayTransaction.escrowList.1.ItemTemplate
2	NOT Tried TO	raiseClaim-BUYER-INR-ITEMBACK-INR_SUCCESS-001		EbayTransaction.escrowList.1.ItemPrice
3	NOT Tried TO	raiseClaim-BUYER-INR-MONEYBACK-INR_SUCCESS-002		Ebay
4	NOT Tried TO	raiseClaim-CS1-NRR-ITEMBACK-INR_SUCCESS-003		
5	NOT Tried TO	raiseClaim-CS1-NRR-ITEMBACK-INR_SUCCESS-004		
6	NOT Tried TO	raiseClaim-CS1-NRR-ITEMBACK-INR_FAILURE-005		
7	NOT Tried TO	raiseClaim-CS1-NRR-ITEMBACK-INR_FAILURE-006		
8	NOT Tried TO	raiseClaim-CS1-NRR-ITEMBACK-INR_BIDNOTC_FAILURE-007		
9	NOT Tried TO	raiseClaim-CS1-NRR-ITEMBACK-INR_BIDNOTC_FAILURE-008		
10	NOT Tried TO	raiseClaim-BUYER-INR-ITEMBACK-INR_AFTER30_FAILURE-009		
11	NOT Tried TO	raiseClaim-BUYER-INR-MONEYBACK-INR_AFTER30_FAILURE-010		
12	NOT Tried TO	raiseClaim-CS1-NRR-ITEMBACK-INR_SUCCESS-011		
13	NOT Tried TO	raiseClaim-CS1-NRR-MONEYBACK-INR_SUCCESS-012		
14	NOT Tried TO	raiseClaim-BUYER-INAD-DIFF_COLOR-REPLACEMENT-SUCCESS-013		
15	NOT Tried TO	raiseClaim-BUYER-INAD-DIFF_COLOR-MONEYBACK-SUCCESS-014		
16	NOT Tried TO	raiseClaim-BUYER-INAD-DIFF_MODEL-REPLACEMENT-SUCCESS-015		
17	NOT Tried TO	raiseClaim-BUYER-INAD-DIFF_MODEL-MONEYBACK-SUCCESS-016		
18	NOT Tried TO	raiseClaim-BUYER-INAD-DIFF_MODEL-MONEYBACK-SUCCESS-017		
19	NOT Tried TO	raiseClaim-BUYER-INAD-DIFF_MODEL-MONEYBACK-SUCCESS-018		
20	NOT Tried TO	raiseClaim-BUYER-INAD-ODOUR-REPLACEMENT-SUCCESS-019		
21	NOT Tried TO	raiseClaim-BUYER-INAD-ODOUR-REPLACEMENT-SUCCESS-020		
22	NOT Tried TO	raiseClaim-BUYER-INAD-ODOUR-MONEYBACK-SUCCESS-021		
23	NOT Tried TO	raiseClaim-BUYER-INAD-ODOUR-SUCCESS-022		
24	NOT Tried TO	raiseClaim-BUYER-INAD-QUALITY-REPLACEMENT-SUCCESS-023		
25	NOT Tried TO	raiseClaim-BUYER-INAD-QUALITY-MONEYBACK-SUCCESS-024		
26	NOT Tried TO	raiseClaim-BUYER-INAD-SEAL-REPLACEMENT-SUCCESS-025		
27	NOT Tried TO	raiseClaim-BUYER-INAD-SEAL-MONEYBACK-SUCCESS-026		
28	NOT Tried TO	raiseClaim-BUYER-INAD-USED-REPLACEMENT-SUCCESS-027		
29	NOT Tried TO	raiseClaim-BUYER-INAD-USED-MONEYBACK-SUCCESS-028		
30	NOT Tried TO	raiseClaim-BUYER-INAD-FAKE-REPLACEMENT-SUCCESS-029		
31	NOT Tried TO	raiseClaim-BUYER-INAD-FAKE-MONEYBACK-SUCCESS-030		
32	NOT Tried TO	raiseClaim-BUYER-INAD-INSTANT-REPLACEMENT-SUCCESS-031		
33	NOT Tried TO	raiseClaim-BUYER-INAD-INSTANT-MONEYBACK-SUCCESS-032		
34	NOT Tried TO	raiseClaim-CS1-INAD-DIFF_COLOR-REPLACEMENT-SUCCESS-033		
35	NOT Tried TO	raiseClaim-CS1-INAD-DIFF_COLOR-MONEYBACK-SUCCESS-034		
36	NOT Tried TO	raiseClaim-CS1-INAD-DIFF_MODEL-REPLACEMENT-SUCCESS-035		
37	NOT Tried TO	raiseClaim-CS1-INAD-DIFF_MODEL-MONEYBACK-SUCCESS-036		
38	NOT Tried TO	raiseClaim-CS1-INAD-DIFF_MODEL-MONEYBACK-SUCCESS-037		
39	NOT Tried TO	raiseClaim-CS1-INAD-EXPIRED-MONEYBACK-SUCCESS-038		
40	NOT Tried TO	raiseClaim-CS1-INAD-EXPIRED-REPLACEMENT-SUCCESS-039		
41	NOT Tried TO	raiseClaim-CS1-INAD-DIFF_PROD-MONEYBACK-SUCCESS-040		
42	NOT Tried TO	raiseClaim-CS1-INAD-ODOUR-REPLACEMENT-SUCCESS-041		
43	NOT Tried TO	raiseClaim-CS1-INAD-ODOUR-MONEYBACK-SUCCESS-042		
44	NOT Tried TO	raiseClaim-CS1-INAD-QUALITY-REPLACEMENT-SUCCESS-043		
45	NOT Tried TO	raiseClaim-CS1-INAD-QUALITY-MONEYBACK-SUCCESS-044		

	AD	AE	AF	AG	AH	AI	AJ
1	EbayTransaction.escrowList.1.CodStatus	EbayTransaction.escrowList.1.EgClaim.ClaimType	EbayTransaction.escrowList.1.EgClaim.PreferredResolution	EbayTransaction.escrowList.1.EgClaim.SellerContactStatus	EscrowDates.creationRange	EscrowDates.deliveredDate	EscrowDates.expectedDeliveryD
2					-10		
3					-10		
4					-10		
5					.9		
6					-3		
7					-3		
8					-3		
9					31		
10					35		
11					48		
12					48		
13					0		
14					0		
15					0		
16					0		
17					0		
18					0		
19					0		
20					0		
21					0		
22					0		
23					0		
24					0		
25					0		
26					0		
27					0		
28					0		
29					0		
30					0		
31					0		
32					0		
33					0		
34					0		
35					0		
36					0		
37					0		
38					0		
39					0		
40					0		
41					0		
42					0		
43					0		
44					0		
45					n		

K10	AU	AK	AL	AM	AN	AO	AP	AQ	
1	te	EscrowDates.expectedDeliveryDate	EscrowDates.shippedDate	EscrowDates.shipBeforeDate	EscrowDates.deliverBeforeDate	EscrowDates.scrubStatus	resultMessage	EbayTransaction.escrowList.0.gClaim.isImageDiscChecked	EbayTransaction.escrowList.0.gClaim.isUploadImage
2				-4	-1	NOT_SHIPPED	INR_SUCCESS	FALSE	FALSE
3				-10	-9	NOT_SHIPPED	INR_SUCCESS	FALSE	FALSE
4				-11	-10	NOT_SHIPPED	INR_SUCCESS	FALSE	FALSE
5				-10	-10	NOT_SHIPPED	INR_SUCCESS	FALSE	FALSE
6						NOT_SHIPPED	INR_BEFORERO_FAILURE	FALSE	FALSE
7						NOT_SHIPPED	INR_BEFORERO_FAILURE	FALSE	FALSE
8						NOT_SHIPPED	INR_BEFORERO_FAILURE	FALSE	FALSE
9						NOT_SHIPPED	INR_BEFORERO_FAILURE	FALSE	FALSE
10						NOT_SHIPPED	INR_AFTER30_FAILURE	FALSE	FALSE
11						NOT_SHIPPED	INR_AFTER30_FAILURE	FALSE	FALSE
12						NOT_SHIPPED	INR_SUCCESS	FALSE	FALSE
13						NOT_SHIPPED	INR_SUCCESS	FALSE	FALSE
14						DELIVERED	SNAAD_DLVRD_SUCCESS	FALSE	TRUE
15						DELIVERED	SNAAD_DLVRD_SUCCESS	FALSE	TRUE
16						DELIVERED	SNAAD_DLVRD_SUCCESS	FALSE	TRUE
17						DELIVERED	SNAAD_DLVRD_SUCCESS	FALSE	TRUE
18						DELIVERED	SNAAD_DLVRD_SUCCESS	FALSE	TRUE
19						DELIVERED	SNAAD_DLVRD_SUCCESS	FALSE	TRUE
20						DELIVERED	SNAAD_DLVRD_SUCCESS	FALSE	TRUE
21						DELIVERED	SNAAD_DLVRD_SUCCESS	FALSE	TRUE
22						DELIVERED	SNAAD_DLVRD_SUCCESS	TRUE	FALSE
23						DELIVERED	SNAAD_DLVRD_SUCCESS	TRUE	FALSE
24						DELIVERED	SNAAD_DLVRD_SUCCESS	TRUE	FALSE
25						DELIVERED	SNAAD_DLVRD_SUCCESS	TRUE	FALSE
26						DELIVERED	SNAAD_DLVRD_SUCCESS	FALSE	TRUE
27						DELIVERED	SNAAD_DLVRD_SUCCESS	FALSE	TRUE
28						DELIVERED	SNAAD_DLVRD_SUCCESS	FALSE	TRUE
29						DELIVERED	SNAAD_DLVRD_SUCCESS	FALSE	TRUE
30						DELIVERED	SNAAD_DLVRD_SUCCESS	FALSE	TRUE
31						DELIVERED	SNAAD_DLVRD_SUCCESS	FALSE	TRUE
32						DELIVERED	SNAAD_DLVRD_SUCCESS	FALSE	TRUE
33						DELIVERED	SNAAD_DLVRD_SUCCESS	FALSE	TRUE
34						DELIVERED	SNAAD_DLVRD_SUCCESS	FALSE	TRUE
35						DELIVERED	SNAAD_DLVRD_SUCCESS	FALSE	FALSE
36						DELIVERED	SNAAD_DLVRD_SUCCESS	FALSE	FALSE
37						DELIVERED	SNAAD_DLVRD_SUCCESS	FALSE	FALSE
38						DELIVERED	SNAAD_DLVRD_SUCCESS	FALSE	FALSE
39						DELIVERED	SNAAD_DLVRD_SUCCESS	FALSE	FALSE
40						DELIVERED	SNAAD_DLVRD_SUCCESS	FALSE	FALSE
41						DELIVERED	SNAAD_DLVRD_SUCCESS	FALSE	FALSE
42						DELIVERED	SNAAD_DLVRD_SUCCESS	FALSE	FALSE
43						DELIVERED	SNAAD_DLVRD_SUCCESS	FALSE	FALSE
44						DELIVERED	SNAAD_DLVRD_SUCCESS	FALSE	FALSE
A1						END_NAVER	ENNA_DLVRD_SUCCESS	EX111	EX111

The screenshot shows the Eclipse IDE interface with the following details:

- Title Bar:** Shows "RaiseClaimServiceTestPlan.java" as the active file.
- Toolbar:** Includes standard Eclipse icons for file operations, search, and refresh.
- Quick Access:** A button for quick access to frequently used resources.
- Java EE:** A button for Java Enterprise Edition tools.
- Debug:** A button for debugging tools.
- Code Area:** The main workspace where the Java code is displayed.
- Left Margin:** Shows line numbers from 1 to 67.
- Right Margin:** Shows status bars for "Writable", "Smart Insert", "1:1", and a vertical ellipsis.

Code Content:

```
1 package com.ebay.moui.test.ebayin.eService;
2
3 import java.lang.reflect.Method;
4 import junit.framework.TestCase;
5 import java.util.LinkedHashMap;
6
7 import org.junit.Test;
8 import org.junit.runner.RunWith;
9 import org.junit.runners.TestNG;
10
11 import com.ebay.moui.component.ebayin.common.eServiceUtil;
12 import com.ebay.moui.component.ebayin.common.EbayTransaction;
13 import com.ebay.moui.component.ebayin.ec.common.EscrubNotes;
14 import com.ebay.moui.component.ebayin.ec.common.UserRole;
15 import com.ebay.moui.component.ebayin.eG.servicePages.RaiseClaimServiceRequest;
16 import com.ebay.moui.component.ebayin.eG.util.EasyFilter;
17 import com.ebay.moui.component.ebayin.eG.util.SubmitClaimRequest;
18 import com.ebay.moui.component.ebayin.servicetools.RaiseClaimServiceConsumer;
19 import com.ebay.moui.component.ebayin.servicetools.SubmitClaimServiceConsumer;
20 import com.ebay.moui.component.ebayin.util.PowerShipUtil;
21 import com.ebay.moui.controller.ContextManager;
22 import com.ebay.moui.controller.EasyFilter;
23 import com.ebay.moui.controller.Logging;
24 import com.ebay.moui.controller.TestPlan;
25 import com.ebay.moui.controller.TestRetryAnalyzer;
26 import com.ebay.moui.util.SpreadSheetUtil;
27 import com.ebay.moui.util.internal.entity.TestObject;
28 import com.ebay.moui.util.internal.type.TestType;
29 import com.ebay.moui.util.internal.type.TestAspect;
30 import com.ebay.moui.util.internal.type.TestType;
31
32 public class RaiseClaimServiceTestPlan extends TestPlan {
33
34     @NotNull private String eGclaimService; private boolean true
35     public static final String TEST_SITE = "moui";
36     protected void checkContext() throws Exception {
37         EasyFilter filter = EasyFilter.equalsIgnoreCase(TestObject.TEST_METHOD,
38             m.getName());
39         filter = EasyFilter.and(filter, EasyFilter.equalsIgnoreCase(
40             TestObject.TEST_SITE,
41             ContextManager.getTestLevelContext((TestContext).getSite())));
42
43         LinkedHashMap<String, Class<?>> classMap = new LinkedHashMap<String, Class<?>>();
44
45         classMap.put("TestObject", TestObject.class);
46         classMap.put("UserRole", UserRole.class);
47         classMap.put("EbayTransaction", EbayTransaction.class);
48         classMap.put("EscrubNotes", EscrubNotes.class);
49
50         return SpreadSheetUtil.getEntitiesFromSpreadsheet(
51             "RaiseClaimServiceTestPlan.class", classMap,
52             "eGclaimsNewIssueService.csv", 0, null, filter);
53     }
54
55     @TestRetryAnalyzer = TestRetryAnalyzer.class, groups =
56     { TestType.Regression, Priorities.P1, TestAspect.Positive,
57       "raiseClaim" }, dataProvider = "eGclaimsService", description = "Raise Claim by CSR and Buyer"
58     public void raiseClaim(String resultMessage, TestObject testObject,
59                           UserRole role, EbayTransaction es, EscrubNotes escrubbles)
60                           throws Exception {
61
62         Logging.info("STEP 1: List items for raise Claim flow");
63         try {
64             PowerShipUtil.createTransaction_v1(es);
65         } catch (Exception e) {
66             e.printStackTrace();
67             throw new Exception("Failed while creating data point", e);
68     }
```

```

54     @Test(retryAnalyzer = TestRetryAnalyzer.class, groups = {
55         TestType.Regression, Priorities.P1, TestAspect.Positive,
56         "raiseClaim" },dataProvider = "eGClaimsService", description = "Raise Claim by (SR1 and Buyer")
57     public void raiseClaim(String escrowId, String resultMessage, TestObject testObject,
58             UserRole role, EbayTransaction es, EscrowDates escrowDates)
59     throws Exception {
60
61     Logging.info("STEP-1: List items for raise claim flow");
62     try {
63         PowerShipUtil.createTransaction_v1(es);
64     } catch (Exception e) {
65         e.printStackTrace();
66         throw new Exception("Failed while creating data point", e);
67     }
68
69     Logging.info("STEP-2: Mark the transaction to deliver");
70     eGUtil.updateTransactionStatus(es.getEscrowList().get(0),
71             es);
72
73     Logging.info("STEP-3: Built the request payload for raise claim");
74     RaiseClaimSvcRequest raiseClaimRequest = new RaiseClaimSvcRequest();
75     if (role.getUserRole().trim().equalsIgnoreCase("SR1")) {
76         raiseClaimRequest.setLoggedInUserName(PowerShipUtil.adminCred
77             .getProperty("eg_crt"));
78     } else {
79         raiseClaimRequest.setLoggedInUserName(es.getBuyer()
80             .getUserName());
81     }
82
83     eGServiceUtil.KaiseClaimRequest(es, raiseClaimRequest);
84     RaiseClaimServiceConsumer reqConsumer = new RaiseClaimServiceConsumer();
85
86     Logging.info("STEP-4: Raise the claim");
87     reqConsumer.raiseClaim(raiseClaimRequest);
88     Logging.info("STEP-5: Validate the claim");
89     eGServiceUtil.KaiseClaimResponseValidation(es);
90
91 }
92
93     @Test(retryAnalyzer = TestRetryAnalyzer.class, enabled = false, groups = {
94         TestType.Regression, Priorities.P1, TestAspect.Positive,
95         "submitClaim" },dataProvider = "eGClaimsService", description = "Submit Claim by Buyer")
96     public void submitClaim(String updateTransactionDate, String deliveryDate,
97             TestObject testObject, UserRole role, EbayTransaction es)
98     throws Exception {
99
100    // List items for raise claim flow
101    try {
102        PowerShipUtil.createTransaction_v1(es);
103    } catch (Exception e) {
104        e.printStackTrace();
105        throw new Exception("Failed while creating data point", e);
106    }
107
108    SubmitClaimRequest submitClaimRequest = new SubmitClaimRequest();
109    eGServiceUtil.submitClaimRequest(es, submitClaimRequest);
110
111    SubmitClaimServiceConsumer submitConsumer = new SubmitClaimServiceConsumer();
112
113    submitConsumer.submitClaim(submitClaimRequest);
114
115    eGServiceUtil.submitClaimResponseValidation(es);
116
117 }
118 }
119 }
120

```

Service:

```

1 package com.ebay.mouli.component.ebayIn.eG.servicePages;
2
3 import javax.xml.bind.annotation.XmlRootElement;
4
5 @XmlRootElement
6 public class RaiseClaimSvcRequest {
7
8     String loggedInUserName; // Mandatory. Max length 25 character
9
10    long escrowId; // Mandatory. Max length 20.
11    long txId; // Mandatory. Max length 20.
12    long itemId; // Mandatory. Max length 20.
13
14    // User selected values
15    String mobileNo; // Mandatory. Max length 15.
16    String landlineNo; // Mandatory. Max length 15.
17
18    int claimType; // Mandatory. Values should be 2/8/9/10
19    String userPreference; // Mandatory. Values should be 1/2/3
20    String remarks; // Mandatory. Max length 4000.
21
22    //Added By Gourav
23    int subSkuId;
24    int buyerInvoiceReceived;
25    int buyerInputImageNotRequired;
26
27    /** Not Contacted, Not Tried = 1
28     * Value for Not Contacted, Tried to = Seller's phone not reachable (S)
29     * Seller does not answer phone (C) Seller does not reply to email (E)
30     * Value Yes = contact by phone (P) / contact by email (E) / contact by both (S)
31
32    int sellerContactOption; // Mandatory
33    int refundPreference; // Mandatory. Values should be 1/2/3/4
34    int buyerMobileVerified;
35
36
37    public String getLoggedInUserName() {
38        return loggedInUserName;
39    }
40    public void setLoggedInUserName(String loggedInUserName) {
41        this.loggedInUserName = loggedInUserName;
42    }
43
44    public long getEscrowId() {
45        return escrowId;
46    }
47    public void setEscrowId(long escrowId) {
48        this.escrowId = escrowId;
49    }
50
51    public long getTxId() {
52        return txId;
53    }
54    public void setTxId(long txId) {
55        this.txId = txId;
56    }
57
58    public long getItemId() {
59        return itemId;
60    }
61    public void setItemId(long itemId) {
62        this.itemId = itemId;
63    }
64
65    public String getMobileNo() {
66        return mobileNo;
67    }

```

```
66     return mobileNo;
67   }
68   public void setMobileNo(String mobileNo) {
69     this.mobileNo = mobileNo;
70   }
71   public String getLandlineNo() {
72     return landlineNo;
73   }
74   public void setLandlineNo(String landlineNo) {
75     this.landlineNo = landlineNo;
76   }
77   public int getClaimType() {
78     return claimType;
79   }
80   public void setClaimType(int claimType) {
81     this.claimType = claimType;
82   }
83   public String getUserPreference() {
84     return userPreference;
85   }
86   public void setUserPreference(String userPreference) {
87     this.userPreference = userPreference;
88   }
89   public String getRemarks() {
90     return remarks;
91   }
92   public void setRemarks(String remarks) {
93     this.remarks = remarks;
94   }
95   public int getSellerContactOption() {
96     return sellerContactOption;
97   }
98   public void setSellerContactOption(int sellerContactOption) {
99     this.sellerContactOption = sellerContactOption;
100  }
101  public int getRefundPreference() {
102    return refundPreference;
103  }
104  public void setRefundPreference(int refundPreference) {
105    this.refundPreference = refundPreference;
106  }
107  public int getSubIssueId() {
108    return subIssueId;
109  }
110  public void setSubIssueId(int subIssueId) {
111    this.subIssueId = subIssueId;
112  }
113  public int getBuyerInvoiceReceived() {
114    return buyerInvoiceReceived;
115  }
116  public void setBuyerInvoiceReceived(int buyerInvoiceReceived) {
117    this.buyerInvoiceReceived = buyerInvoiceReceived;
118  }
119  public int getBuyerInputImageNotRequired() {
120    return buyerInputImageNotRequired;
121  }
122  public void setBuyerInputImageNotRequired(int buyerInputImageNotRequired) {
123    this.buyerInputImageNotRequired = buyerInputImageNotRequired;
124  }
125  public void setClaimType(int claimType) {
126    this.claimType = claimType;
127  }
128  public String getUserPreference() {
129    return userPreference;
130  }
131  public void setUserPreference(String userPreference) {
132    this.userPreference = userPreference;
133  }
134  public String getRemarks() {
135    return remarks;
136  }
137  public void setRemarks(String remarks) {
138    this.remarks = remarks;
139  }
140  public int getSellerContactOption() {
141    return sellerContactOption;
142  }
143  public void setSellerContactOption(int sellerContactOption) {
144    this.sellerContactOption = sellerContactOption;
145  }
146  public int getRefundPreference() {
147    return refundPreference;
148  }
149  public void setRefundPreference(int refundPreference) {
150    this.refundPreference = refundPreference;
151  }
152  public int getSubIssueId() {
153    return subIssueId;
154  }
155  public void setSubIssueId(int subIssueId) {
156    this.subIssueId = subIssueId;
157  }
158  public int getBuyerInvoiceReceived() {
159    return buyerInvoiceReceived;
160  }
161  public void setBuyerInvoiceReceived(int buyerInvoiceReceived) {
162    this.buyerInvoiceReceived = buyerInvoiceReceived;
163  }
164  public int getBuyerInputImageNotRequired() {
165    return buyerInputImageNotRequired;
166  }
167  public void setBuyerInputImageNotRequired(int buyerInputImageNotRequired) {
168    this.buyerInputImageNotRequired = buyerInputImageNotRequired;
169  }
170  public int getBuyerMobileVerified() {
171    return buyerMobileVerified;
172  }
173  public void setBuyerMobileVerified(int buyerMobileVerified) {
174    this.buyerMobileVerified = buyerMobileVerified;
175  }
176 }
```

```
82  public void setClaimType(int claimType) {
83    this.claimType = claimType;
84  }
85  public String getUserPreference() {
86    return userPreference;
87  }
88  public void setUserPreference(String userPreference) {
89    this.userPreference = userPreference;
90  }
91  public String getRemarks() {
92    return remarks;
93  }
94  public void setRemarks(String remarks) {
95    this.remarks = remarks;
96  }
97  public int getSellerContactOption() {
98    return sellerContactOption;
99  }
100  public void setSellerContactOption(int sellerContactOption) {
101    this.sellerContactOption = sellerContactOption;
102  }
103  public int getRefundPreference() {
104    return refundPreference;
105  }
106  public void setRefundPreference(int refundPreference) {
107    this.refundPreference = refundPreference;
108  }
109  public int getSubIssueId() {
110    return subIssueId;
111  }
112  public void setSubIssueId(int subIssueId) {
113    this.subIssueId = subIssueId;
114  }
115  public int getBuyerInvoiceReceived() {
116    return buyerInvoiceReceived;
117  }
118  public void setBuyerInvoiceReceived(int buyerInvoiceReceived) {
119    this.buyerInvoiceReceived = buyerInvoiceReceived;
120  }
121  public int getBuyerInputImageNotRequired() {
122    return buyerInputImageNotRequired;
123  }
124  public void setBuyerInputImageNotRequired(int buyerInputImageNotRequired) {
125    this.buyerInputImageNotRequired = buyerInputImageNotRequired;
126  }
127  public int getBuyerMobileVerified() {
128    return buyerMobileVerified;
129  }
130  public void setBuyerMobileVerified(int buyerMobileVerified) {
131    this.buyerMobileVerified = buyerMobileVerified;
132  }
133  public void setClaimType(int claimType) {
134    this.claimType = claimType;
135  }
136  public String getUserPreference() {
137    return userPreference;
138  }
139  public void setUserPreference(String userPreference) {
140    this.userPreference = userPreference;
141  }
142  public String getRemarks() {
143    return remarks;
144  }
145  public void setRemarks(String remarks) {
146    this.remarks = remarks;
147  }
148  public int getSellerContactOption() {
149    return sellerContactOption;
150  }
151  public void setSellerContactOption(int sellerContactOption) {
152    this.sellerContactOption = sellerContactOption;
153  }
154  public int getRefundPreference() {
155    return refundPreference;
156  }
157  public void setRefundPreference(int refundPreference) {
158    this.refundPreference = refundPreference;
159  }
160  public int getSubIssueId() {
161    return subIssueId;
162  }
163  public void setSubIssueId(int subIssueId) {
164    this.subIssueId = subIssueId;
165  }
166  public int getBuyerInvoiceReceived() {
167    return buyerInvoiceReceived;
168  }
169  public void setBuyerInvoiceReceived(int buyerInvoiceReceived) {
170    this.buyerInvoiceReceived = buyerInvoiceReceived;
171  }
172  public int getBuyerInputImageNotRequired() {
173    return buyerInputImageNotRequired;
174  }
175  public void setBuyerInputImageNotRequired(int buyerInputImageNotRequired) {
176    this.buyerInputImageNotRequired = buyerInputImageNotRequired;
177  }
178  public int getBuyerMobileVerified() {
179    return buyerMobileVerified;
180  }
181  public void setBuyerMobileVerified(int buyerMobileVerified) {
182    this.buyerMobileVerified = buyerMobileVerified;
183  }
184 }
```

Screenshot of Eclipse IDE showing the Java EE perspective. The central view displays the code for `RaiseClaimServiceTestPI`. The code implements the `eGServiceUtil` interface and extends `RaiseClaimSvcResponse`. It includes methods for setting transaction details and claim numbers, and handling errors.

```

package com.ebay.mauli.component.ebayIn.eG.servicePages;

import java.util.ArrayList;
import javax.xml.bind.annotation.XmlRootElement;
import com.ebay.mauli.component.billing.payments.ebay.utils.api.pgw.request;
import com.ebay.mauli.component.billing.payments.ebay.utils.api.pgw.response;
import com.ebay.mauli.component.billing.payments.ebay.utils.soa;
import com.ebay.mauli.component.billing.payments.ebay.utils.soa.billingport;
import com.ebay.mauli.component.billing.payments.ebay.utils.soa.pgwservi;
import com.ebay.mauli.component.billing.payments.ebay.utils.soa.pgwservi;
import com.ebay.mauli.component.bizpolicy;
import com.ebay.mauli.component.checkout;
import com.ebay.mauli.component.cookiemanager;
import com.ebay.mauli.component.coupon;
import com.ebay.mauli.component.cs;
import com.ebay.mauli.component.dct;
import com.ebay.mauli.component.ebayIn.batchPages;
import com.ebay.mauli.component.ebayIn.book;
import com.ebay.mauli.component.ebayIn.books.db;
import com.ebay.mauli.component.ebayIn.browse;
import com.ebay.mauli.component.ebayIn.common;
import com.ebay.mauli.component.ebayIn.coupons;
import com.ebay.mauli.component.ebayIn.eG.common;
import com.ebay.mauli.component.ebayIn.eG.pages;
import com.ebay.mauli.component.ebayIn.eG.servicePages;
import com.ebay.mauli.component.ebayIn.eG.util;
import com.ebay.mauli.component.ebayIn.fashion.configClasses;
import com.ebay.mauli.component.ebayIn.fashion.pages;
import com.ebay.mauli.component.ebayIn.fashion.util;

import com.ebay.mauli.component.billing.payments.ebay.utils.api.pgw.request;
import com.ebay.mauli.component.billing.payments.ebay.utils.api.pgw.response;
import com.ebay.mauli.component.billing.payments.ebay.utils.soa;
import com.ebay.mauli.component.billing.payments.ebay.utils.soa.billingport;
import com.ebay.mauli.component.billing.payments.ebay.utils.soa.pgwservi;
import com.ebay.mauli.component.billing.payments.ebay.utils.soa.pgwservi;
import com.ebay.mauli.component.bizpolicy;
import com.ebay.mauli.component.checkout;
import com.ebay.mauli.component.cookiemanager;
import com.ebay.mauli.component.coupon;
import com.ebay.mauli.component.cs;
import com.ebay.mauli.component.dct;
import com.ebay.mauli.component.ebayIn.batchPages;
import com.ebay.mauli.component.ebayIn.book;
import com.ebay.mauli.component.ebayIn.books.db;
import com.ebay.mauli.component.ebayIn.browse;
import com.ebay.mauli.component.ebayIn.common;
import com.ebay.mauli.component.ebayIn.coupons;
import com.ebay.mauli.component.ebayIn.eG.common;
import com.ebay.mauli.component.ebayIn.eG.pages;
import com.ebay.mauli.component.ebayIn.eG.servicePages;
import com.ebay.mauli.component.ebayIn.eG.util;
import com.ebay.mauli.component.ebayIn.fashion.configClasses;
import com.ebay.mauli.component.ebayIn.fashion.pages;
import com.ebay.mauli.component.ebayIn.fashion.util;

public class RaiseClaimSvcResponse {
    long claimReqNo;
    AckValue ack = AckValue.FAILURE;
    private List<ClaimServiceError> errors;
    public List<ClaimServiceError> getErrors() {
        return errors;
    }
    @XmlElement(name="transactionList")
    @XmlElement(name="transactions")
    List<TransactionDetails> transactionList = new ArrayList<TransactionDetails>();
    public List<TransactionDetails> getTransactionDetails() {
        return transactionList;
    }
    public AckValue getAck() {
        return ack;
    }
    public void setAck(AckValue failure) {
        this.ack = failure;
    }
    public long getClaimReqNo() {
        return claimReqNo;
    }
    public void setClaimReqNo(long claimReqNo) {
        this.claimReqNo = claimReqNo;
    }
}

```

Screenshot of Eclipse IDE showing the Java EE perspective. The central view displays the code for `AckValue`. This enum defines various status codes such as SUCCESS, FAILURE, PARTIAL, and PARTIAL_FAILURE. It also includes a static method `fromValue` for converting strings to enum values.

```

package com.ebay.mauli.component.ebayIn.eG.servicePages;

import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement
public enum AckValue {
    SUCCESS("Success"),
    FAILURE("Failure"),
    PARTIAL("Partial"),
    PARTIAL_FAILURE("PartialFailure");
    private final String value;
    AckValue(String v) {
        value = v;
    }
    public String value() {
        return value;
    }
    public static AckValue fromValue(String v) {
        for (AckValue c : AckValue.values()) {
            if (c.value.equals(v)) {
                return c;
            }
        }
        throw new IllegalArgumentException(v);
    }
}

```

Project Explorer Navigator Type Hierarchy

```

1 package com.ebay.mauli.component.ebayIn.eG.servicePages;
2
3 import java.util.ArrayList;
4
5 @XmlRootElement
6 public class ActionStateMappingResponse {
7     private AckValue ack = AckValue.FAILURE;
8
9     public AckValue getAck() {
10         return ack;
11     }
12
13     public void setAck(AckValue failure) {
14         this.ack = failure;
15     }
16
17     @XmlElement(name = "errors")
18     private List<Error> errors = new ArrayList<Error>();
19
20     public List<Error> getErrors() {
21         return errors;
22     }
23
24     @XmlElement(name = "stateMaster")
25     @XmlElement(name = "stateMaster")
26     private List<ActionStateMappingVO> actionStateMappingList = new ArrayList<ActionStateMappingVO>;
27
28
29     public List<ActionStateMappingVO> getAttributesList() {
30         return actionStateMappingList;
31     }
32
33 }
34
35
36
37
38
39
40
41
42
43

```

Markers Properties Servers Data Source Explore Snippets Console Progress Results of running c Package Explorer Call Hierarchy

No operations to display at this time.

Writable Smart Insert 1:1

Project Explorer Navigator Type Hierarchy

```

1 package com.ebay.mauli.component.ebayIn.eG.servicePages;
2
3 import com.sun.xml.txw2.annotation.XmlElement;
4
5 @XmlElement
6 public class ActionStateMappingVO {
7     private long mappingId;
8     private long claimStatusId;
9     private long issueTypeId;
10    private long actionId;
11
12    public long getMappingId() {
13        return mappingId;
14    }
15
16    public void setMappingId(long mappingId) {
17        this.mappingId = mappingId;
18    }
19
20    public long getClaimStatusId() {
21        return claimStatusId;
22    }
23
24    public void setClaimStatusId(long claimStatusId) {
25        this.claimStatusId = claimStatusId;
26    }
27
28    public long getIssueTypeId() {
29        return issueTypeId;
30    }
31
32    public void setIssueTypeId(long issueTypeId) {
33        this.issueTypeId = issueTypeId;
34    }
35
36    public long getActionId() {
37        return actionId;
38    }
39
40    public void setActionId(long actionId) {
41        this.actionId = actionId;
42    }
43
44
45
46
47
48
49 }

```

Markers Properties Servers Data Source Explore Snippets Console Progress Results of running c Package Explorer Call Hierarchy

No operations to display at this time.

Writable Smart Insert 1:1

Screenshot of Eclipse IDE showing the Java EE perspective. The Project Explorer view shows a large number of Java files under the package com.ebay.maul.component.ebayIn.eG.servicePages. The current file is AckValue.java, which contains the following code:

```

1 package com.ebay.maul.component.ebayIn.eG.servicePages;
2
3 public class ActionTypeVO {
4     private int actionTypeId;
5     private String actionType;
6     private int actionStatus;
7     private String dateOfResolution;
8     private String actionStatusName;
9     private String expiryDate;
10
11    public String getExpiryDate() {
12        return expiryDate;
13    }
14
15    public void setExpiryDate(String expiryDate) {
16        this.expiryDate = expiryDate;
17    }
18
19    public String getActionStatusName() {
20        return actionStatusName;
21    }
22
23    public void setActionStatusName(String actionstatusName) {
24        this.actionStatusName = actionstatusName;
25    }
26
27    public String getDateOfResolution() {
28        return dateOfResolution;
29    }
30
31    public void setDateOfResolution(String dateOfResolution) {
32        this.dateOfResolution = dateOfResolution;
33    }
34
35    public int getActionStatus() {
36        return actionStatus;
37    }
38
39    public void setActionStatus(int actionStatus) {
40        this.actionStatus = actionStatus;
41    }
42
43    public int getActionTypeId() {
44        return actionTypeId;
45    }
46
47    public void setActionTypeId(int actionTypeId) {
48        this.actionTypeId = actionTypeId;
49    }
50
51    public String getActionType() {
52        return actionType;
53    }
54
55    public void setActionType(String actionPerformed) {
56        this.actionType = actionPerformed;
57    }
58
59    public int getActionStatusC() {
60        return actionStatus;
61    }
62
63    public void setActionStatusC(int actionStatus) {
64        this.actionStatus = actionStatus;
65    }
66
67    public int getActionTypeIdC() {
68        return actionTypeId;
69    }
70
71    public void setActionTypeIdC(int actionTypeId) {
72        this.actionTypeId = actionTypeId;
73    }
74
75    public String getActionTypeC() {
76        return actionType;
77    }
78
79    public void setActionTypeC(String actionPerformed) {
80        this.actionType = actionPerformed;
81    }
82
83    public int getActionStatusO() {
84        return actionStatus;
85    }
86
87    public void setActionStatusO(int actionStatus) {
88        this.actionStatus = actionStatus;
89    }
90
91    public int getActionTypeIdO() {
92        return actionTypeId;
93    }
94
95    public void setActionTypeIdO(int actionTypeId) {
96        this.actionTypeId = actionTypeId;
97    }
98
99    public String getActionTypeO() {
100        return actionType;
101    }
102
103    public void setActionTypeO(String actionPerformed) {
104        this.actionType = actionPerformed;
105    }
106}

```

The Java EE perspective toolbar is visible at the top, and the status bar at the bottom indicates "Writable" and "Smart Insert".

Screenshot of Eclipse IDE showing the Java EE perspective. The Project Explorer view shows a large number of Java files under the package com.ebay.maul.component.ebayIn.eG.servicePages. The current file is ClaimServiceError.java, which contains the following code:

```

1 package com.ebay.maul.component.ebayIn.eG.servicePages;
2
3 import com.sun.xml.txw2.annotation.XmlElement;
4
5 @XmlElement
6 public class ClaimServiceError extends Error {
7
8     public static final ClaimServiceError VALIDATION_ERROR = new ClaimServiceError(2001, "Please provide valid input details");
9     public static final ClaimServiceError CSR_VALIDATION_ERROR = new ClaimServiceError(2002, "CSRF token occurred");
10    public static final ClaimServiceError DUPLICATE_RECORD_ERROR = new ClaimServiceError(2003, "Duplicate record claim already raised");
11    public static final ClaimServiceError RAISE_CLAIM_ERROR = new ClaimServiceError(2004, "Error occurred while raising the claim");
12
13    public static final ClaimServiceError SMS_SEND_ERROR = new ClaimServiceError(2005, "Error occurred while sending SMS");
14    public static final ClaimServiceError CSC_CLAIM_MAP_UPDATE_ERROR = new ClaimServiceError(2006, "Error occurred while updating CSR claim map");
15
16    public static final ClaimServiceError ERR_MSG_INR_CLAIM_RAISED_WITHIN_SEVEN_DAYS = new ClaimServiceError(2007, "You cannot raise an item not received claim within 7 days of transaction.");
17    public static final ClaimServiceError ERR_MSG_CLAIM_RAISED_AFTER_THIRTY_DAYS_OF_DELIVERY = new ClaimServiceError(2008, "A claim cannot be raised after 30 days from the date of delivery.");
18    public static final ClaimServiceError ERR_MSG_INR_CLAIM_RAISED_AFTER_THIRTY_DAYS = new ClaimServiceError(2009, "Claim cannot be raised after 30 days from the date of transaction.");
19
20    public ClaimServiceError() {
21
22    }
23
24    /**
25     * @param errorCode
26     * @param msg
27     */
28    protected ClaimServiceError(int errorCode, String msg) {
29        this.setErrorCode(errorCode);
30        this.setErrorMsg(msg);
31    }
32
33}

```

The Java EE perspective toolbar is visible at the top, and the status bar at the bottom indicates "Writable" and "Smart Insert".

```

1 package com.ebay.mout.component.ebayIn.eG.servicePages;
2
3 import java.util.Date;
4 import java.util.List;
5
6 import com.ebay.moui.component.ebayIn.serviceUtils.RemarksDetails;
7
8 public class EgClaimHelperUtil {
9
10    /**
11     * @param detail
12     * @return
13     */
14    public static String getTransData(TransactionDetails detail) {
15        List<RemarksDetails> list = detail.getCrRemarks();
16        int action = 0;
17        String remark = "";
18        int actionStatus = 0;
19        long remarkId = -99;
20
21        if (list == null || list.isEmpty()) remark = detail.getUserRemark();
22        if (list != null && !list.isEmpty()) {
23            RemarksDetails rem = list.get(0);
24
25            if (rem.getActionStatus() == 2)
26                action = 0;
27            else
28                action = rem.getActionId();
29            actionStatus = rem.getActionStatus();
30            detail.setLastActionId(action);
31
32            if (rem.getActionStatus() == 2)
33                actionStatus = 0;
34            else
35                actionStatus = rem.getActionId();
36            detail.setLastActionId(action);
37
38            if (rem.getExpDate() != null)
39                expire = rem.getExpDate();
40            remark = "";
41            remarkId = rem.getRemarkId();
42            /*
43             * ) else remarkId = -99;
44             */
45        }
46
47        return detail.getClaimType() + DELIMITER + detail.getBuyerPreference() + DELIMITER + detail.getStatus() + getStatusId() + DELIMITER + action
48        + DELIMITER + expire + DELIMITER + remarkId + DELIMITER + actionStatus + DELIMITER + remark + DELIMITER + detail.toString() + DELIMITER+detail.getCrSeqNo();
49    }
50
51    /**
52     * @param detail
53     * @return
54     */
55    public static ActionTypeVO getActionVOForTransaction(TransactionDetails detail) {
56        ActionTypeVO actionVO = new ActionTypeVO();
57        List<RemarksDetails> list = detail.getCrRemarks();
58        int action = 0;
59        int actionStatus = 0;
60        if (list != null && !list.isEmpty())
61        {
62            RemarksDetails rem = list.get(0);
63
64            if (rem.getActionStatus() == 2)
65                action = 0;
66            else
67                action = rem.getActionId();
68            actionStatus = rem.getActionStatus();
69            detail.setLastActionId(action);
70
71            if (rem.getExpDate() != null)
72                expire = rem.getExpDate();
73            remark = "";
74            remarkId = rem.getRemarkId();
75            /*
76             * ) else remarkId = -99;
77             */
78
79        }
80
81        return detail.getClaimType() + DELIMITER + detail.getBuyerPreference() + DELIMITER + detail.getStatus() + getStatusId() + DELIMITER + action
82        + DELIMITER + expire + DELIMITER + remarkId + DELIMITER + actionStatus + DELIMITER + remark + DELIMITER + detail.toString() + DELIMITER+detail.getCrSeqNo();
83    }
84
85    /**
86     * @param detail
87     * @return
88     */
89    public static ActionTypeVO getActionVOForTransaction(TransactionDetails detail) {
90        ActionTypeVO actionVO = new ActionTypeVO();
91        List<RemarksDetails> list = detail.getCrRemarks();
92        int action = 0;
93        int actionStatus = 0;
94        if (list != null && !list.isEmpty())
95        {
96            RemarksDetails rem = list.get(0);
97
98            if (rem.getActionStatus() == 2)
99                action = 0;
100           actionStatus = rem.getActionStatus();
101           actionVO.setDateOfResolution(CommonUtil.FormatDateForClaims(rem.getExpDate()));
102           actionVO.setXpiryDate(CommonUtil.FormatDate(rem.getExpDate()));
103
104        }
105        return actionVO;
106    }
107
108 }

```

```

1 package com.ebay.mout.component.ebayIn.eG.servicePages;
2
3 import java.util.Date;
4 import java.util.List;
5
6 import com.ebay.moui.component.ebayIn.serviceUtils.RemarksDetails;
7
8 public class EgClaimHelperUtil {
9
10    /**
11     * @param detail
12     * @return
13     */
14    public static String getTransData(TransactionDetails detail) {
15        List<RemarksDetails> list = detail.getCrRemarks();
16        int action = 0;
17        String remark = "";
18        int actionStatus = 0;
19        long remarkId = -99;
20
21        if (list == null || list.isEmpty())
22        {
23            RemarksDetails rem = list.get(0);
24
25            if (rem.getActionStatus() == 2)
26                action = 0;
27            else
28                action = rem.getActionId();
29            actionStatus = rem.getActionStatus();
30            detail.setLastActionId(action);
31
32            if (rem.getExpDate() != null)
33                expire = rem.getExpDate();
34            remark = "";
35            remarkId = rem.getRemarkId();
36            /*
37             * ) else remarkId = -99;
38             */
39        }
40
41        return detail.getClaimType() + DELIMITER + detail.getBuyerPreference() + DELIMITER + detail.getStatus() + getStatusId() + DELIMITER + action
42        + DELIMITER + expire + DELIMITER + remarkId + DELIMITER + actionStatus + DELIMITER + remark + DELIMITER + detail.toString() + DELIMITER+detail.getCrSeqNo();
43    }
44
45    /**
46     * @param detail
47     * @return
48     */
49    public static ActionTypeVO getActionVOForTransaction(TransactionDetails detail) {
50        ActionTypeVO actionVO = new ActionTypeVO();
51        List<RemarksDetails> list = detail.getCrRemarks();
52        int action = 0;
53        int actionStatus = 0;
54        if (list != null && !list.isEmpty())
55        {
56            RemarksDetails rem = list.get(0);
57
58            if (rem.getActionStatus() == 2)
59                action = 0;
60            else
61                action = rem.getActionId();
62            actionStatus = rem.getActionStatus();
63            detail.setLastActionId(action);
64
65            if (rem.getExpDate() != null)
66                expire = rem.getExpDate();
67            remark = "";
68            remarkId = rem.getRemarkId();
69            /*
70             * ) else remarkId = -99;
71             */
72
73        }
74
75        return actionVO;
76    }
77
78    /**
79     * @param detail
80     * @return
81     */
82    public static ActionTypeVO getActionVOForTransaction(TransactionDetails detail) {
83        ActionTypeVO actionVO = new ActionTypeVO();
84        List<RemarksDetails> list = detail.getCrRemarks();
85        int action = 0;
86        int actionStatus = 0;
87        if (list != null && !list.isEmpty())
88        {
89            RemarksDetails rem = list.get(0);
90
91            if (rem.getActionStatus() == 2)
92                action = 0;
93            else
94                action = rem.getActionId();
95            actionStatus = rem.getActionStatus();
96            actionVO.setDateOfResolution(CommonUtil.FormatDateForClaims(rem.getExpDate()));
97            actionVO.setXpiryDate(CommonUtil.FormatDate(rem.getExpDate()));
98
99        }
100        return actionVO;
101    }
102
103    /**
104     * @param detail
105     * @return
106     */
107    public static ActionTypeVO getActionVOForTransaction(TransactionDetails detail) {
108        ActionTypeVO actionVO = new ActionTypeVO();
109        List<RemarksDetails> list = detail.getCrRemarks();
110        int action = 0;
111        int actionStatus = 0;
112        if (list != null && !list.isEmpty())
113        {
114            RemarksDetails rem = list.get(0);
115
116            if (rem.getActionStatus() == 2)
117                action = 0;
118            else
119                action = rem.getActionId();
120            actionStatus = rem.getActionStatus();
121            actionVO.setDateOfResolution(CommonUtil.FormatDateForClaims(rem.getExpDate()));
122            actionVO.setXpiryDate(CommonUtil.FormatDate(rem.getExpDate()));
123
124        }
125        return actionVO;
126    }
127
128 }

```

Screenshot of Eclipse IDE showing the Java EE perspective. The Project Explorer view shows a large number of Java files under the package com.ebay.mauli.component.ebayIn.eG.servicePages. The code editor displays the Error.java file, which contains the following code:

```

1 package com.ebay.mauli.component.ebayIn.eG.servicePages;
2
3 import com.sun.xml.txw2.annotation.XmlElement;
4
5 @XmlElement
6 public class Error {
7
8     int errorCode;
9     String errorMessage;
10
11     public static final Error INVALID_PARAM = new Error(1001, "Please provide valid input details");
12     public static final Error APP_ERROR = new Error(1002, "Application Encountered Error. Please try after some time");
13     // code for submit claim
14     public static final Error INVALID_TXN = new Error(1003, "Error while fetching transaction details");
15     public static final Error DUPLICATE_TXN = new Error(1002, "Claim has already been created for this transaction.");
16
17     public Error() {
18
19     }
20
21     protected Error(int errorCode, String msg) {
22         super();
23         this.errorCode = errorCode;
24         this.errorMessage = msg;
25     }
26
27     public int getErrorCode() {
28
29     }
30     public void setErrorCode(int errorCode) {
31         this.errorCode = errorCode;
32     }
33     public String getErrorMessage() {
34         return errorMessage;
35     }
36     public void setErrorMessage(String errorMessage) {
37         this.errorMessage = errorMessage;
38     }
}

```

The code editor interface includes tabs for ActionStateMappingVO.java, ActionTypeVO.java, EgClaimHelperUtil.java, ClaimServiceError.java, and Error.java. The status bar at the bottom shows 'Writable' and 'Smart Insert'.

Screenshot of Eclipse IDE showing the Java EE perspective. The Project Explorer view shows a subset of Java files under the same package structure. The code editor displays the IsVerifiedOTP.java file, which contains the following code:

```

1 package com.ebay.mauli.component.ebayIn.eG.servicePages;
2
3 public class IsVerifiedOTP {
4
5     // As handle IsVerifiedOTP service output
6     private boolean verified;
7
8     public boolean isVerified() {
9         return verified;
10    }
11
12    public void setVerified(boolean verified) {
13        this.verified = verified;
14    }
15
16
17}

```

The code editor interface includes tabs for ActionStateMappingResponse.java, ActionTypeVO.java, ClaimServiceError.java, EgClaimHelperUtil.java, Error.java, and IsVerifiedOTP.java. The status bar at the bottom shows 'Writable' and 'Smart Insert'.

The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer** (left): Shows a large tree of Java files under the package `com.ebay.maul.component`. The tree includes sub-packages like `billing`, `payments`, `ebay`, `utils`, and `pgw`.
- Editor View** (center): Displays the Java code for the class `EgClaimHelperUtil`. The code is annotated with Javadoc and includes methods for setting and getting various parameters like `ack`, `allowClaimSubmit`, `errorMessage`, `token`, and `verified`.
- Bottom Status Bar**: Shows tabs for `Markers`, `Properties`, `Servers`, `Data Source Explore`, `Snippets`, `Console`, `Progress`, and `Results of running c`.

The screenshot shows an IDE interface with multiple tabs at the top, including "ActionStateMappingV", "ActionTypeVO.java", "EgClaimHelperUtil.java", "ClaimServiceError.java", "Error.java", "IsVerifiedOTP.java", "RaiseClaimSvcReqe", "SendOTP.java", and "ShippingDetailsHist". The main pane displays the Java code for the "ActionStateMappingV" class. The code is annotated with JBoss Seam annotations such as `@Name`, `@Scope("view")`, `@In`, `@Out`, and `@RequestValue`. It also includes annotations from the Seam Persistence framework like `@Entity`, `@Table`, `@Id`, `@Column`, and `@Version`. The code defines various methods for interacting with a database table, including getters and setters for columns like `SHIPPING_HISTORY_ID`, `CARRIER_ID`, `AWB_NUMBER`, `PICKUP_DATE`, `PICKUP_DATE`, `PACKAGE_ID`, `EBAY_SHIPPED`, `SHIPPING_FEES`, `EG_SHIPPING_FEES`, and `LAST_MODIFIED_BY`.

```
1 package com.ebay.moui.component.ebayin.eig.servicePoges;
2
3 import java.util.Date;
4
5 //@@com.lantougroup.provider("shippingservice")
6 public interface ShippingdetailsHistory extends DaoDO {
7
8     public long getShippingHistoryId();
9
10    public void setShippingHistoryId(long shippingHistoryId);
11
12    //@Column(name = "SHIPPING_DETAILS_ID")
13    public long getShippingDetailsId();
14
15    public void setShippingDetailsId(long shippingDetailsId);
16
17    //@Column(name = "CARRIER_ID")
18    public int getCarrierId();
19
20    public void setCarrierId(int carrierId);
21
22    //@Column(name = "CARRIER_NAME")
23    public String getCarrierName();
24
25    public void setCarrierName(String carrierName);
26
27    //@Column(name = "AWB_NUMBER")
28    public String getAwbNumber();
29
30    public void setAwbNumber(String awbNumber);
31
32    //@Column(name = "PICKUP_DATE")
33    public Date getPickupDate();
34
35    public void setPickupDate(Date pickupDate);
36
37    public int getStatus();
38
39    public void setStatus(int status);
40
41    //@Column(name = "PACKAGE_ID")
42    public String getPackageId();
43
44    public void setPackageId(String packageId);
45
46    //@Column(name = "EBAY_SHIPPED")
47    public int getEbayShipped();
48
49    public void setEbayShipped(int ebayShipped);
50
51    //@Column(name = "SHIPPING_FEES")
52    public double getShippingFees();
53
54    public void setShippingFees(double shippingFees);
55
56    //@Column(name = "EG_SHIPPING_FEES")
57    public double getEgShippingFees();
58
59    public void setEgShippingFees(double egShippingFees);
60
61    //@Column(name = "LAST_MODIFIED_BY")
62    public String getLastModifiedBy();
63
64    public void setLastModifiedBy(String lastModifiedBy);
65
66    public String getRemarks();
67
68}
```

The screenshot shows the Java code for `ActionStateMappingV` in an IDE. The code is annotated with various annotations such as `@Column`, `@Import`, and `@Column(name = "LAST_MODIFIED_BY")`. The code defines methods for setting and getting shipping history ID, shipping details ID, carrier ID, carrier name, AWB number, pickup date, status, package ID, eBay shipped status, shipping fees, and last modified by.

```
11  public void setShippingHistoryId(long shippingHistoryId);
12  // @Column(name = "SHIPPING_DETAILS_ID")
13  public long getShippingDetailsId();
14
15  public void setShippingDetailsId(long shippingDetailsId);
16
17  // @Column(name = "CARRIER_ID")
18  public int getCarrierId();
19
20  public void setCarrierId(int carrierId);
21
22  // @Column(name = "CARRIER_NAME")
23  public String getCarrierName();
24
25  public void setCarrierName(String carrierName);
26
27  // @Column(name = "AWB_NUMBER")
28  public String getAwbNumber();
29
30  public void setAwbNumber(String awbNumber);
31
32  // @Column(name = "PICKUP_DATE")
33  public Date getPickupDate();
34
35  public void setPickupDate(Date pickupDate);
36
37  public int getStatus();
38
39  public void setStatus(int status);
40
41  // @Column(name = "PACKAGE_ID")
42  public String getPackageId();
43
44  public void setPackageId(String packageId);
45
46  // @Column(name = "EBAY_SHIPPED")
47  public int getEbayShipped();
48
49  public void setEbayShipped(int ebayShipped);
50
51
52  // @Column(name = "SHIPPING_FEES")
53  public double getShippingFees();
54
55  public void setShippingFees(double shippingFees);
56
57  // @Column(name = "EG_SHIPPING_FEES")
58  public double getEgShippingFees();
59
60  public void setEgShippingFees(double egShippingFees);
61
62  // @Column(name = "LAST_MODIFIED_BY")
63  public String getLastModifiedBy();
64
65  public void setLastModifiedBy(String lastModifiedBy);
66
67  public String getRemarks();
68
69  public void setRemarks(String remarks);
70
71  // @Column(name = "CREATION_DATE")
72  public Date getCreationDate();
73
74  public void setCreationDate(Date creationDate);
75 }
76 }
```

The screenshot shows the Java code for `Status` in an IDE. The code uses a static map to map status IDs to their corresponding values and display values. It includes methods for getting the status list, status ID by type, status value, display value, and status details.

```
1  package com.ebay.mouli.component.ebayIn.ee.servicePages;
2
3  import java.util.HashMap;
4
5  public class Status
6
7  {
8
9      public static enum StatusType {OPEN, IN_PROGRESS, CLOSED, REOPENED, ACCEPTED, REJECTED};
10
11     private static Map<String, Status> statusMap = new HashMap<String, Status>();
12     private static Map<Integer, Status> statusIdMap = new HashMap<Integer, Status>();
13     private static Map<Integer, Set<Status>> statusMapping = new HashMap<Integer, Set<Status>>();
14
15     static
16     {
17         populateStatusMap();
18     }
19
20     private int statusId;
21     private String statusVal;
22     private String statusDisplayVal;
23
24     public Status(int statusId, String statusVal, String statusDisplayVal)
25     {
26         this.statusId = statusId;
27         this.statusVal = statusVal;
28         this.statusDisplayVal = statusDisplayVal;
29     }
30
31     /* public static List<Status> getStatusList()
32     {
33         List<Status> list = new ArrayList<Status>();
34         Set<keys> keys = statusMap.keySet();
35         Iterator<keys> iter = keys.iterator();
36         while (iter.hasNext())
37         {
38             Status s1s = statusMap.get(iter.next());
39             list.add(s1s);
40         }
41         return list;
42     }*/
43
44     public static Set<Status> getClaimStatusList(int stateId)
45     {
46         return statusMapping.get(stateId);
47     }
48
49     public int getStatusId()
50     {
51         return this.statusId;
52     }
53
54     public static int getStatusId(StatusType type)
55     {
56         return statusMap.get(type.toString()).getStatusId();
57     }
58
59     public String getStatusValue()
60     {
61         return this.statusVal;
62     }
63
64     public String getStatusDisplayValue()
65     {
66         return this.statusDisplayVal;
67     }
68
69     public static Status getStatusDetails(int id)
70     {
71     }
```

The screenshot shows the Java code for ActionStateMappingVO.java in an IDE. The code defines a static map of Status objects and methods to populate it from a database. It includes methods for getting status details by type, converting to string, and comparing objects.

```
70     {
71         return statusMap.get(id);
72     }
73
74     public static Status getStatusDetails(StatusType type)
75     {
76         return statusMap.get(type.toString());
77     }
78
79     public String toString()
80     {
81         return this.statusDisplayVal;
82     }
83
84
85     public static void populateStatusMap()
86     {
87         try{
88             List<EGStateMaster> stateValues = EgStateMasterDAO.getInstance().findAllByType(StateTypeEnum.CLAIM_STATUS.getId(), ReadSets.FULL);
89             for(EGStateMaster state : stateValues){
90                 statusMap.put(state.getStateValue(), new Status((int)state.getId(),state.getStateValue(),state.getStateDisplayValue()));
91                 statusMap.put((int)state.getId(), new Status((int)state.getId(),state.getStateValue(),state.getStateDisplayValue()));
92             }
93             populateStatusMapping();
94         }catch(FinderException e){
95             //logger.log(Level.SEVERE,"Unable to load the claim status values",e);
96             throw new Error("Unable to load the claim status values");
97         }
98     }
99
100    private static void populateStatusMapping(){
101        Open set = new HashSet<Status>();
102        Set<Status> inProgressSet = new LinkedHashSet<Status>();
103        Set<Status> closedSet = new LinkedHashSet<Status>();
104        Set<Status> reopenedSet = new LinkedHashSet<Status>();
105        Set<Status> acceptedSet = new LinkedHashSet<Status>();
106        Set<Status> rejectedSet = new LinkedHashSet<Status>();
107
108        openSet.add(statusMap.get(StatusType.IN_PROGRESS.toString()));
109        openSet.add(statusMap.get(StatusType.CLOSED.toString()));
110        inProgressSet.add(statusMap.get(StatusType.IN_PROGRESS.toString()));
111        inProgressSet.add(statusMap.get(StatusType.CLOSED.toString()));
112        closedSet.add(statusMap.get(StatusType.IN_PROGRESS.toString()));
113        closedSet.add(statusMap.get(StatusType.CLOSED.toString()));
114        reopenedSet.add(statusMap.get(StatusType.REOPENED.toString()));
115        reopenedSet.add(statusMap.get(StatusType.ACCEPTED.toString()));
116        reopenedSet.add(statusMap.get(StatusType.REJECTED.toString()));
117        acceptedSet.add(statusMap.get(StatusType.ACCEPTED.toString()));
118        acceptedSet.add(statusMap.get(StatusType.REJECTED.toString()));
119        acceptedSet.add(statusMap.get(StatusType.REJECTING.toString()));
120        rejectedSet.add(statusMap.get(StatusType.ACCEPTED.toString()));
121        rejectedSet.add(statusMap.get(StatusType.REJECTED.toString()));
122        rejectedSet.add(statusMap.get(StatusType.REJECTING.toString()));
123
124        statusMapping.put(statusMap.get(StatusType.OPEN.toString()),openSet);
125        statusMapping.put(statusMap.get(StatusType.IN_PROGRESS.toString()),inProgressSet);
126        statusMapping.put(statusMap.get(StatusType.CLOSED.toString()),closedSet);
127        statusMapping.put(statusMap.get(StatusType.REOPENED.toString()),reopenedSet);
128        statusMapping.put(statusMap.get(StatusType.ACCEPTED.toString()),acceptedSet);
129        statusMapping.put(statusMap.get(StatusType.REJECTING.toString()),rejectedSet);
130    }
131
132    public boolean equals(Object obj){
133        if(obj != null){
134            Status statusObj = (Status)obj;
135            if(statusObj.statusId == this.statusId && statusObj.statusVal.equals(this.statusVal))
136                return true;
137        }
138        return false;
139    }
140
141 }
```

The screenshot shows the same Java code for ActionStateMappingVO.java in the IDE, but with significant portions of the code removed or commented out. The removed sections include the static map population logic and the comparison method.

```
77     }
78
79     public String toString()
80     {
81         return this.statusDisplayVal;
82     }
83
84
85     public static void populateStatusMap()
86     {
87         try{
88             List<EGStateMaster> stateValues = EgStateMasterDAO.getInstance().findAllByType(StateTypeEnum.CLAIM_STATUS.getId(), ReadSets.FULL);
89             for(EGStateMaster state : stateValues){
90                 statusMap.put(state.getStateValue(), new Status((int)state.getId(),state.getStateValue(),state.getStateDisplayValue()));
91                 statusMap.put((int)state.getId(), new Status((int)state.getId(),state.getStateValue(),state.getStateDisplayValue()));
92             }
93             populateStatusMapping();
94         }catch(FinderException e){
95             //logger.log(Level.SEVERE,"Unable to load the claim status values",e);
96             throw new Error("Unable to load the claim status values");
97         }
98     }
99
100    private static void populateStatusMapping(){
101        Open set = new HashSet<Status>();
102        Set<Status> inProgressSet = new LinkedHashSet<Status>();
103        Set<Status> closedSet = new LinkedHashSet<Status>();
104        Set<Status> reopenedSet = new LinkedHashSet<Status>();
105        Set<Status> acceptedSet = new LinkedHashSet<Status>();
106        Set<Status> rejectedSet = new LinkedHashSet<Status>();
107
108        openSet.add(statusMap.get(StatusType.IN_PROGRESS.toString()));
109        openSet.add(statusMap.get(StatusType.CLOSED.toString()));
110        inProgressSet.add(statusMap.get(StatusType.IN_PROGRESS.toString()));
111        inProgressSet.add(statusMap.get(StatusType.CLOSED.toString()));
112        closedSet.add(statusMap.get(StatusType.IN_PROGRESS.toString()));
113        closedSet.add(statusMap.get(StatusType.CLOSED.toString()));
114        reopenedSet.add(statusMap.get(StatusType.REOPENED.toString()));
115        reopenedSet.add(statusMap.get(StatusType.ACCEPTED.toString()));
116        reopenedSet.add(statusMap.get(StatusType.REJECTED.toString()));
117        acceptedSet.add(statusMap.get(StatusType.ACCEPTED.toString()));
118        acceptedSet.add(statusMap.get(StatusType.REJECTED.toString()));
119        acceptedSet.add(statusMap.get(StatusType.REJECTING.toString()));
120        rejectedSet.add(statusMap.get(StatusType.ACCEPTED.toString()));
121        rejectedSet.add(statusMap.get(StatusType.REJECTED.toString()));
122        rejectedSet.add(statusMap.get(StatusType.REJECTING.toString()));
123
124        statusMapping.put(statusMap.get(StatusType.OPEN.toString()),openSet);
125        statusMapping.put(statusMap.get(StatusType.IN_PROGRESS.toString()),inProgressSet);
126        statusMapping.put(statusMap.get(StatusType.CLOSED.toString()),closedSet);
127        statusMapping.put(statusMap.get(StatusType.REOPENED.toString()),reopenedSet);
128        statusMapping.put(statusMap.get(StatusType.ACCEPTED.toString()),acceptedSet);
129        statusMapping.put(statusMap.get(StatusType.REJECTING.toString()),rejectedSet);
130    }
131
132    public boolean equals(Object obj){
133        if(obj != null){
134            Status statusObj = (Status)obj;
135            if(statusObj.statusId == this.statusId && statusObj.statusVal.equals(this.statusVal))
136                return true;
137        }
138        return false;
139    }
140
141 }
```

The screenshot shows a Java code editor with the file `SubmitClaimRequest.java` open. The code defines a class `SubmitClaimRequest` with various fields and methods for managing claims. The code includes annotations like `@XmlElement` and `@XmlAttribute`. The interface `SubmitClaimRequest` is implemented by `ActionStateMappingVO.java` and `ActionTypeVO.java`.

```
1 package com.ebay.moui.component.ebayIn.eg.servicePages;
2
3 import javax.xml.bind.annotation.XmlRootElement;
4
5
6 @XmlRootElement
7 public class SubmitClaimRequest {
8
9     long transactionId;
10    long buyerId;
11    long sellerId;
12    long itemId;
13    long escrowId;
14    String claimType;
15    String remarks;
16
17
18    public long getTransactionId() {
19        return transactionId;
20    }
21    public void setTransactionId(long transactionId) {
22        this.transactionId = transactionId;
23    }
24    public long getBuyerId() {
25        return buyerId;
26    }
27    public void setBuyerId(long buyerId) {
28        this.buyerId = buyerId;
29    }
30    public long getSellerId() {
31        return sellerId;
32    }
33    public void setSellerId(long sellerId) {
34        this.sellerId = sellerId;
35    }
36    public long getItemId() {
37        return itemId;
38    }
39    public void setItemId(long itemId) {
40        this.itemId = itemId;
41    }
42    public long getEscrowId() {
43        return escrowId;
44    }
45    public void setEscrowId(long escrowId) {
46        this.escrowId = escrowId;
47    }
48    public String getClaimType() {
49        return claimType;
50    }
51    public void setClaimType(String claimType) {
52        this.claimType = claimType;
53    }
54    public String getRemarks() {
55        return remarks;
56    }
57    public void setRemarks(String remarks) {
58        this.remarks = remarks;
59    }
60}
61}
```

The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer:** Shows a large tree of Java packages under the com.ebay.maul component. Notable packages include billing.payments, component.billing.payments, component.billing, component.checkout, component.cookieManager, component.coupon, component.cs, component.dct, component.ebayIn, component.ebayIn.batchPages, component.ebayIn.books, component.ebayIn.books.db, component.ebayIn.browser, component.common, component.ebayIn.common, component.ebayIn.coupons, component.ebayIn.coupons.db, component.ebayIn.daily, component.ebayIn.ebayIn, component.ebayIn.eG, component.ebayIn.eG.pages, and component.ebayIn.eG.servicePages.
- Editor:** Displays the `EgClaimHelperUt` class from the `com.ebay.maul.component.ebayIn.eG.servicePages` package. The code implements the `SubmitClaimResponse` interface, handling XML input and output for errors and errors.
- Toolbar:** Standard Eclipse toolbar with icons for file operations, search, and preferences.
- Top Bar:** Shows "Writable", "Smart Insert", and "1:1".
- Right Side:** Shows the Java EE perspective with a tree view of the `Submit` package containing `ack`, `error`, `getAck`, `getErrors`, and `getErrorsList`.
- Bottom:** Eclipse status bar with tabs for Markers, Properties, Servers, Data Source Explore, Snippets, Console, Progress, Results of running c, Package Explorer, Call Hierarchy, and Help.

```

1 package com.ebay.maul.component.ebayin.eG.servicePages;
2
3 import java.text.DateFormat;
4
5 /**
6 * @author rishabh
7 */
8
9 public class TransactionDetails {
10
11     /* Data related to claim_txn table Starts.. */
12
13     private long m_sellerId;
14     private String m_buyerId;
15     private long m_buyerInternalId;
16     private String m_buyerSite;
17     private long m_itemId;
18     private long m_itemInternalId;
19     private String m_listingFormat;
20     private long m_listedPayId;
21     private long m_transactionId;
22     private long m_transactionInternalId;
23     private String m_paymentMode;
24     private String m_sellerId;
25     private long m_sellerInternalId;
26
27     private double m_couponAmt;
28     private double m_itemAmount;
29     private double m_mitPoid;
30     /* Capturing amount paid using payback */
31     private double m_paybackAmount;
32     // Capture amount paid on cod fee
33     private double m_codConvenienceFee;
34     private Status m_status;
35     private String m_userRemark;
36     private String m_itemTitle;
37     private int m_buyerPreference;
38
39     private Date creationDate;
40     private Date claimCloseDate;
41     private Date lastModifiedDate;
42     long m_escrowId;
43     private int m_xnType;
44     private long m_variationId;
45
46     private int psShipment;
47
48     // Buyer Details
49     private String m_BuyerMobileNumber;
50     private String m_BuyerLandlineNumber;
51     private String m_BuyerEmailId;
52
53     // Seller Details
54     private String m_SellerEmailId;
55     private int m_SellerContactId;
56     private String m_SellerProtection;
57     private String m_Buyer.First_Name;
58     private String m_Buyer.Last_Name;
59     private String m_Seller.First_Name;
60     private String m_Seller.Last_Name;
61
62     private String sellerContactNumber;
63     private Date sellerProtectionClaimOpenedDate;
64
65 }

```

```

1 package com.ebay.maul.component.ebayin.eG.servicePages;
2
3 public class ValidateOTP {
4
5     /* To handle validateOTP service output */
6     private boolean otpValidation;
7     private boolean allowClaimSubmit;
8
9     public boolean isOtpValidation() {
10         return otpValidation;
11     }
12
13     public void setOtpValidation(boolean otpValidation) {
14         this.otpValidation = otpValidation;
15     }
16
17     public boolean isAllowClaimSubmit() {
18         return allowClaimSubmit;
19     }
20
21     public void setAllowClaimSubmit(boolean allowClaimSubmit) {
22         this.allowClaimSubmit = allowClaimSubmit;
23     }
24 }

```

The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer:** Shows packages like com.ebay.maul.component.ebayIn.browse, com.ebay.maul.component.ebayIn.common, com.ebay.maul.component.ebayIn.coupons, com.ebay.maul.component.ebayIn.coupons.db, com.ebay.maul.component.ebayIn.daily, com.ebay.maul.component.ebayIn.eG.common, com.ebay.maul.component.ebayIn.eG.pages, com.ebay.maul.component.ebayIn.eG.servicePages, com.ebay.maul.component.ebayIn.eg.servicePages, com.ebay.maul.component.ebayIn.eg.util, com.ebay.maul.component.ebayIn.eg.util.Assert, com.ebay.maul.component.ebayIn.eg.util.ActionStateMappingResponse, com.ebay.maul.component.ebayIn.eg.util.ActionStateMappingVO, com.ebay.maul.component.ebayIn.eg.util.ActionTypeVO, com.ebay.maul.component.ebayIn.eg.util.ClaimServiceError, com.ebay.maul.component.ebayIn.eg.util.EgClaimHelperUtil, com.ebay.maul.component.ebayIn.eg.util.Error, com.ebay.maul.component.ebayIn.eg.util.IsVerifiedOTP, com.ebay.maul.component.ebayIn.eg.util.RaiseClaimSvRequest, com.ebay.maul.component.ebayIn.eg.util.RaiseClaimSvResponse, com.ebay.maul.component.ebayIn.eg.util.SendOTP, com.ebay.maul.component.ebayIn.eg.util.ShippingDetailsHistory, com.ebay.maul.component.ebayIn.eg.util.Status, com.ebay.maul.component.ebayIn.eg.util.SubmitClaimRequest, com.ebay.maul.component.ebayIn.eg.util.SubmitClaimResponse, com.ebay.maul.component.ebayIn.eg.util.TransactionDetails, com.ebay.maul.component.ebayIn.eg.util.ValidateOTP, com.ebay.maul.component.ebayIn.eG.util, com.ebay.maul.component.ebayIn.eG.util.EGDEUtil, com.ebay.maul.component.ebayIn.eG.util.eGUtil, com.ebay.maul.component.ebayIn.eG.util.eGResultingMessage.properties, com.ebay.maul.component.ebayIn.fashion.configClasses, com.ebay.maul.component.ebayIn.fashion.pages, com.ebay.maul.component.ebayIn.fashion.util, com.ebay.maul.component.ebayIn.finders, com.ebay.maul.component.ebayIn.geb, com.ebay.maul.component.ebayIn.geb.pages, com.ebay.maul.component.ebayIn.LOCOTool, com.ebay.maul.component.ebayIn.mobile.pages, com.ebay.maul.component.ebayIn.mobile.util, com.ebay.maul.component.ebayIn.paisapay, com.ebay.maul.component.ebayIn.payback.refund, com.ebay.maul.component.ebayIn.prod.pages, com.ebay.maul.component.ebayIn.prod.resources, com.ebay.maul.component.ebayIn.ps, com.ebay.maul.component.ebayIn.ps.lnIntegration, and com.ebay.maul.component.ebayIn.ps.adm.
- Code Editor:** The main editor window displays the Java code for `ActionStateMapper.java`. The code includes imports for various Java packages, utility classes, and annotations like `@WebService` and `@WebMethod`. It defines several methods, including `getActionState`, `getClaimStatus`, `getClaimType`, `getEgStatus`, `getRefundPreference`, `getSellerProtectionStatus`, `getSubIssue`, `getUserNotification`, and `getUserRole`.
- Outline View:** Located at the bottom left, it shows the outline of the current file, including class definitions and method signatures.
- Java EE Debug:** A small window on the right side of the interface.

The screenshot shows the Eclipse IDE interface with the following details:

- Title Bar:** Shows multiple open tabs including "ActionStateMappingVO.java", "ActionTypeVO.java", "ClaimServiceError.java", "eGResultingMessage.properties", "CheckClaimTestPlan.java", and "eGUtl.java".
- Quick Access:** A button in the top right corner.
- Java EE:** A button in the top right corner.
- Code Editor:** The main area displays the Java code for "ActionStateMappingVO.java". The code includes imports for various eBay components and utility classes, as well as static fields and methods. It also contains annotations like "@XmlRootElement" and "@XmlElement".
- Sidebar:** On the left, there's a file tree showing project structure. On the right, there are several toolbars and a status bar at the bottom.

```
1  import com.ebay.mouli.component.ebayIn.ec.common.Assertable;
2  import com.ebay.mouli.component.ebayIn.ec.pages.BuyerRaiseClaimHomePage;
3  import com.ebay.mouli.component.ebayIn.ec.pages.RaiseClaimHomePage;
4  import com.ebay.mouli.component.ebayIn.ec.pages.RaiseClaimPage;
5  import com.ebay.mouli.component.ebayIn.ec.util.Assert;
6  import com.ebay.mouli.component.ebayIn.ec.ServicePages.IsVerifiedOTP;
7  import com.ebay.mouli.component.ebayIn.ps.PowerShipLabels.ShippingCarrier;
8  import com.ebay.mouli.component.ebayIn.ps.IpIntegration.PSTracking;
9  import com.ebay.mouli.component.ebayIn.servicemanager.Consumer;
10 import com.ebay.mouli.component.ebayIn.servicemanager.IConsumer;
11 import com.ebay.mouli.component.ebayInitils.EbayInitilsModule;
12 import com.ebay.mouli.component.ebayInitils.PowerShipUtil;
13 import com.ebay.mouli.component.ebayInitils.ReturnTransactionDetailsPage;
14 import com.ebay.mouli.component.page.SignIn.SignInPage;
15 import com.ebay.mouli.controller.Assertion;
16 import com.ebay.mouli.controller.Logging;
17 import com.ebay.mouli.driver.WebDriver;
18 import com.ebay.mouli.driver.WebElement.BuyerLoginPage;
19 import com.ebay.mouli.driver.WebElement.WebPage;
20 import com.ebay.mouli.helper.ThreadHelper;
21 import com.ebay.mouli.helper.WebDriverHelper;
22 import com.ebay.mouli.util.CSVUtil;
23 import com.ebay.mouli.util.DBUtil;
24 import com.ebay.mouli.util.InternalEntity.PhoneNumberType;
25 import com.ebay.mouli.util.InternalEntity.User;
26 import com.ebay.powership.ipinteg.notification.EbayAWBStatusEnum;
27 import com.ebay.soap.eBLBaseComponents.AddressType;
28 import com.ebay.soap.eBLBaseComponents.GetItemTransactionsResponseType;
29 import com.ebay.soap.eBLBaseComponents.TransactionArrayType;
30 import com.ebay.soap.eBLBaseComponents.TransactionType;
31 import com.sun.jersey.api.client.Client;
32 import com.sun.jersey.api.client.WebResource;
33
34 public class eGUtl {
35
36     private static int shipmentActionsArray[] = { 3, 7, 21, 30, 31 };
37     private static int returnShipmentActionIds[] = { 3, 30 };
38     private static int replacementShipmentActionIds[] = { 21, 31 };
39     private static SimpleDateFormat dbDateFormat = new SimpleDateFormat(
40         "yyyy-MM-dd HH:mm:ss");
41     private static SimpleDateFormat dbDateFormat2 = new SimpleDateFormat(
42         "yy-MM-dd HH:mm:ss");
43     private static SimpleDateFormat dbDateFormat3 = new SimpleDateFormat(
44         "yyyy-MM-dd");
45
46     // added by aspin
47     public static Properties egCred = new Properties();
48
49     static {
50         try {
51             Class<?> clz = Class
52                 .forName("com.ebay.mouli.component.ebayIn.eGUtl");
53             egCred.load(clz.getResourceAsStream("eGResultingMessage.properties"));
54         } catch (Exception e) {
55             // TODO Auto-generated catch block
56             e.printStackTrace();
57         }
58     }
59
60     public static void raiseClaimNegativeTest(EbayEscrow escrow, User user,
61         String isNegativeTest, String errorMessage) throws Exception {
62
63         EGclaim egclaim = escrow.getEGclaim();
64         RaiseClaimHomePage pRaiseClaimHomePage = null;
65         if (user != null)
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
```

```

123     pRaiseClaimHomePage = new BuyerRaiseClaimHomePage(
124         user.getLoginName(), "password");
125     else
126         pRaiseClaimHomePage = new RaiseClaimHomePage(
127             PowerSightUtil.admIncred.getProperty("eg_csr1"), "password");
128
129     RaiseClaimPage pRaiseClaimPage = pRaiseClaimHomePage;
130
131     // add phone number
132     pRaiseClaimPage.lnkBuyerMobileEdit.click();
133     pRaiseClaimPage.waitForElementEditable(pRaiseClaimPage.txtBuyerMobile);
134     pRaiseClaimPage.txtBuyerMobile.clear();
135     if (user.getBuyerMobile().toCharArray().length != 0) {
136         pRaiseClaimPage.txtBuyerMobile.sendKeys(user.getBuyerMobile());
137     }
138
139     pRaiseClaimPage.linkBuyerLandlineEdit.click();
140     pRaiseClaimPage.waitForElementEditable(pRaiseClaimPage.linkBuyerLandlineEdit);
141     pRaiseClaimPage.txtBuyerLandline.clear();
142     if (user.getBuyerLandline().toCharArray().length != 0) {
143         pRaiseClaimPage.txtBuyerLandline.sendKeys(user.getBuyerLandline());
144     }
145
146     if (egClaim.getClaimType() != null)
147         pRaiseClaimPage.chooseClaimType(egClaim.getClaimType());
148     if (egClaim.getPreferredResolution() != null)
149         pRaiseClaimPage.chooseBuyerPref(egClaim.getPreferredResolution());
150     if (egClaim.getRefundPreference() != null)
151         pRaiseClaimPage.setRefundPreference(egClaim
152             .getRefundPreference());
153     if (egClaim.getBuyerContactStatus() != null)
154         pRaiseClaimPage.addAddressInfo(egClaim.getBuyerContactStatus());
155     if (egClaim.getBuyerRemarks() != null)
156         pRaiseClaimPage.enterComments(egClaim.getBuyerRemarks());
157
158     pRaiseClaimPage.btnSaveSubmit.click();
159     Alert alert = null;
160     if (!isNsgpitiveTest.equalsIgnoreCase("TRUE")) {
161         alert = pRaiseClaimPage.getAlert();
162         Assert.assertEquals(alert.getText(), egUtil.eGred.getMessage(),
163             "There is a mismatch of " + uiMessage);
164     }
165 }
166
167 public static RaiseClaimSuccessPage raiseClaim(EbayEscrow escrow, User user)
168     throws Exception {
169
170     EGClaim egClaim = escrow.getEgClaim();
171     RaiseClaimHomePage pRaiseClaimHomePage = null;
172     if (user != null)
173         pRaiseClaimHomePage = new BuyerRaiseClaimHomePage(
174             user.getLoginName(), "password");
175     else
176         pRaiseClaimHomePage = new RaiseClaimHomePage(
177             PowerSightUtil.admIncred.getProperty("eg_csr1"), "password");
178
179     return raiseClaim(pRaiseClaimHomePage, escrow.getPaisoPayId(), egClaim,
180         user);
181 }
182
183 public static synchronized RaiseClaimSuccessPage raiseClaim(
184     RaiseClaimHomePage pRaiseClaimHomePage, String paisoPayId,
185     EGClaim egClaim, User user) throws Exception {
186
187     RaiseClaimPage pRaiseClaimPage = pRaiseClaimHomePage
188         .viewOrderDetails(paisoPayId);
189

```

```

190     // add Phone number
191     if (user != null)
192         if (pRaiseClaimPage.txtBuyerMobile.getAttribute("value") == null
193             || pRaiseClaimPage.txtBuyerMobile.getAttribute("value")
194                 .equals("")) {
195             pRaiseClaimPage.lnkBuyerMobileEdit.click();
196             pRaiseClaimPage.waitForElementEditable(pRaiseClaimPage.txtBuyerMobile);
197             pRaiseClaimPage.txtBuyerMobile.sendKeys(user.getBuyerMobile());
198         }
199
200     if (pRaiseClaimPage.txtBuyerLandline.getAttribute("value") == null
201         || pRaiseClaimPage.txtBuyerLandline.getAttribute("value")
202             .equals("")) {
203         pRaiseClaimPage.lnkBuyerLandlineEdit.click();
204         pRaiseClaimPage.waitForElementEditable(pRaiseClaimPage.lnkBuyerLandlineEdit);
205         pRaiseClaimPage.txtBuyerLandline.clear();
206         pRaiseClaimPage.txtBuyerLandline.sendKeys(user.getBuyerLandline());
207     }
208
209 } else {
210     pRaiseClaimPage.lnkBuyerMobileEdit.click();
211     pRaiseClaimPage.waitForElementEditable(pRaiseClaimPage.txtBuyerMobile);
212     pRaiseClaimPage.txtBuyerMobile.clear();
213     pRaiseClaimPage.txtBuyerMobile.sendKeys(egUtil.eGred
214         .getProperty("BUYER_MOBILE"));
215
216     pRaiseClaimPage.lnkBuyerLandlineEdit.click();
217     pRaiseClaimPage.waitForElementEditable(pRaiseClaimPage.lnkBuyerLandlineEdit);
218     pRaiseClaimPage.txtBuyerLandline.clear();
219     pRaiseClaimPage.txtBuyerLandline.sendKeys(egUtil.eGred
220         .getProperty("BUYER_LN"));
221 }
222
223 pRaiseClaimPage.chooseClaimType(egClaim.getClaimType());
224
225 // Added by Gaurav
226 // For subtypes
227 pRaiseClaimPage.chooseSubIssue(egClaim.getSubIssue());
228
229 // for invoice
230 /*
231     * if(egClaim.getReceiveInvoice()==true){
232     * pRaiseClaimPage.rbnYesReceivedInvoice.click(); } else{
233     * pRaiseClaimPage.rbnNotReceivedInvoice.click(); }
234     */
235
236 pRaiseClaimPage.chooseBuyerPref(egClaim.getPreferredResolution());
237 pRaiseClaimPage.selectRefundPreference(egClaim.getRefundPreference());
238
239 // Added by Gaurav
240 if (egClaim.getClaimType().toString().equals("ITEM_NOT_AS_EXPECTED"))
241     if (egClaim.getClaimType().toString()
242         .equals("ITEM_NOT_WORKING_BROKEN")
243         || egClaim.getClaimType().toString()
244             .equals("PART_OF_THE_ORDER_MISSING")) {
245
246     try {
247         if (egClaim.getClaimType().toString()
248             .equals("ITEM_NOT_AS_EXPECTED")
249             || egClaim.isUploadImage() == true) {
250
251         File file = new File(".");
252         pRaiseClaimPage.txtImageBrowse.sendKeys(file
253             .getCanonicalPath());

```

```

257     + "> " + pRaiseClaimPage.btnAddImage.click();
258     ThreadHelper.waitForSeconds(30); // update
259     Assertion.assertEquals(pRaiseClaimPage.btnAddImage.isDisplayed(),
260                           false);
261     Assertion.assertEquals(pRaiseClaimPage.btnDelete.isDisplayed(),
262                           true);
263     "Image Delete button is not displaying");
264 }
265 if (eClaim.getClaimTypeO().toString()
266     .equals("ITEM NOT WORKING/BROKEN") || eClaim
267     .getClaimTypeO().toString()
268     .equals("PART_OF_THE_ORDER_MISSING"))
269     && eClaim.isImageIsChecked() == true) {
270     pRaiseClaimPage.chkBoundaryImage.click();
271 }
272 } catch (Exception e) {
273     e.printStackTrace();
274 }
275 }
276 pRaiseClaimPage.btnAddInfo(eClaim.getSellerContactStatusO);
277 pRaiseClaimPage.enterComments(eClaim.getBuyerRemarksO);
278 return pRaiseClaimPage.submitClaim();
279 }
280 }
281 }
282 }
283 }
284 // Added by GANESH
285 public static RaiseClaimSuccessPage raiseClaimNew(EboyEscrow escrow,
286     User user, UserRole userRole) throws Exception {
287     EGClaim eClaim = escrow.getEgClaim();
288     RaiseClaimHomePage pRaiseClaimHomePage = null;
289     if (user.getUserRoleO().equals("CSA1"))
290         pRaiseClaimHomePage = new RaiseClaimHomePage(
291             PowerShipUtil.admincred.getProperty("eg_csr1"), "password");
292     // return raiseClaimNew(pRaiseClaimHomePage, "366000394", eClaim,
293     // user, userRole);
294     else
295         pRaiseClaimHomePage = new BuyerRaiseClaimHomePage(
296             user.getLoginName(), "password");
297     // pRaiseClaimHomePage = new BuyerRaiseClaimHomePage("testgsb1",
298     // "password");
299     return pRaiseClaimNew(pRaiseClaimHomePage, escrow.getPssoPayIDO(),
300     eClaim, user, userRole);
301 }
302 }
303 public static synchronized RaiseClaimSuccessPage raiseClaimNew(
304     RaiseClaimHomePage pRaiseClaimHomePage, String pssoPayId,
305     EGClaim eClaim, User user, UserRole userRole) throws Exception {
306     RaiseClaimPage pRaiseClaimPage = pRaiseClaimHomePage
307         .viewOrderDetails(pssoPayId);
308     // add Phone number
309     String mobileNumber = user.getUserIdO();
310     if (mobileNumber != null) {
311         if (pRaiseClaimPage.txtBuyerMobile.getAttribute("value") == null
312             || pRaiseClaimPage.txtBuyerMobile.getAttribute("value")
313             .equals("")) {
314             pRaiseClaimPage.lnkBuyerMobileEdit.click();
315             pRaiseClaimPage
316                 .waitForElementVisible(pRaiseClaimPage.txtBuyerMobile);
317             pRaiseClaimPage.txtBuyerMobile.sendKeys(user.getBuyerPhone());
318         }
319     }
320     pRaiseClaimPage
321         .waitForElementVisible(pRaiseClaimPage.txtBuyerLandline);
322     pRaiseClaimPage.txtBuyerLandline.sendKeys(user.getBuyerPhone());
323     pRaiseClaimPage
324         .waitForElementVisible(pRaiseClaimPage.txtBuyerMobile);
325     pRaiseClaimPage.txtBuyerMobile.sendKeys(user.getBuyerPhone());
326 }
327 if (pRaiseClaimPage.txtBuyerLandline.getAttribute("value") == null
328     || pRaiseClaimPage.txtBuyerLandline.getAttribute("value")
329     .equals("")) {
330     pRaiseClaimPage.lnkBuyerLandlineEdit.click();
331     pRaiseClaimPage
332         .waitForElementVisible(pRaiseClaimPage.lnkBuyerLandlineEdit);
333     pRaiseClaimPage.txtBuyerLandline
334         .sendKeys(user.getBuyerPhone());
335 }
336 }
337 if (pRaiseClaimPage.lnkBuyerMobile.getAttribute("value") == null
338     || pRaiseClaimPage.lnkBuyerMobile.getAttribute("value")
339     .equals("")) {
340     pRaiseClaimPage
341         .waitForElementVisible(pRaiseClaimPage.lnkBuyerMobileEdit);
342     pRaiseClaimPage.txtBuyerMobile.clear();
343     pRaiseClaimPage.txtBuyerMobile.sendKeys(user.getBuyerPhone());
344     pRaiseClaimPage
345         .waitForElementVisible(pRaiseClaimPage.lnkBuyerLandlineEdit);
346     pRaiseClaimPage.txtBuyerLandline
347         .waitForElementVisible(pRaiseClaimPage.lnkBuyerLandlineEdit);
348     pRaiseClaimPage.txtBuyerLandline.clear();
349     pRaiseClaimPage.txtBuyerLandline.sendKeys(user.getBuyerPhone());
350     pRaiseClaimPage
351         .waitForElementVisible(pRaiseClaimPage.txtBuyerMobileEdit);
352 }
353 ThreadHelper.waitForSeconds(2);
354 pRaiseClaimPage.chooseClaimType(eClaim.getClaimTypeO());
355 }
356 // for subIssue
357 ThreadHelper.waitForSeconds(2);
358 pRaiseClaimPage.chooseSubIssue(eClaim.getSubIssueO());
359 if (eClaim.getSubIssueListO() == null) {
360     String subIssueItems[] = eClaim.getSubIssueListO().split(",");
361     for (int i = 0; i < subIssueItems.length; i++) {
362         subIssueItems[i] = SubIssue.valueOf(subIssueItems[i])
363             .getDescription();
364     }
365 }
366 pRaiseClaimPage.lblSubIssueType.click();
367 Assertion
368     .assertTrue(pRaiseClaimPage
369     .verifySubIssueDropDownList(subIssueItems),
370     "subIssue list is not expected as per the selected Issue Type");
371 }
372 }
373 // Invoice
374 if (eClaim.isInvoiceReceivedO().toLowerCase().equals("no")) {
375     Assertion.assertFalse(pRaiseClaimPage.lblInvoice.isDisplayed(),
376                           "Invoice Radio button should not be displayed");
377 }
378 }
379 }
380 }
381 }
382 }
383 }
384 }
385 }
386 }
387 }
388 }
389 }
390 }
391 }
392 }
393 }
394 }
395 }
396 }
397 }
398 }
399 }
400 }
401 }
402 }
403 }
404 }
405 }
406 }
407 }
408 }
409 }
410 }
411 }
412 }
413 }
414 }
415 }
416 }
417 }
418 }
419 }
420 }
421 }
422 }
423 }
424 }
425 }
426 }
427 }
428 }
429 }
430 }
431 }
432 }
433 }
434 }
435 }
436 }
437 }
438 }
439 }
440 }
441 }
442 }
443 }
444 }
445 }
446 }
447 }
448 }
449 }
450 }
451 }
452 }
453 }
454 }
455 }
456 }
457 }
458 }
459 }
460 }
461 }
462 }
463 }
464 }
465 }
466 }
467 }
468 }
469 }
470 }
471 }
472 }
473 }
474 }
475 }
476 }
477 }
478 }
479 }
480 }
481 }
482 }
483 }
484 }
485 }
486 }
487 }
488 }
489 }
490 }
491 }
492 }
493 }
494 }
495 }
496 }
497 }
498 }
499 }
500 }
501 }
502 }
503 }
504 }
505 }
506 }
507 }
508 }
509 }
510 }
511 }
512 }
513 }
514 }
515 }
516 }
517 }
518 }
519 }
520 }
521 }
522 }
523 }
524 }
525 }
526 }
527 }
528 }
529 }
530 }
531 }
532 }
533 }
534 }
535 }
536 }
537 }
538 }
539 }
540 }
541 }
542 }
543 }
544 }
545 }
546 }
547 }
548 }
549 }
550 }
551 }
552 }
553 }
554 }
555 }
556 }
557 }
558 }
559 }
560 }
561 }
562 }
563 }
564 }
565 }
566 }
567 }
568 }
569 }
570 }
571 }
572 }
573 }
574 }
575 }
576 }
577 }
578 }
579 }
580 }
581 }
582 }
583 }
584 }
585 }
586 }
587 }
588 }
589 }
590 }
591 }
592 }
593 }
594 }
595 }
596 }
597 }
598 }
599 }
599 }

```

```

324     .waitForElementVisible(pRaiseClaimPage.txtBuyerMobile);
325     pRaiseClaimPage.txtBuyerMobile.sendKeys(user.getBuyerPhone());
326 }
327 if (pRaiseClaimPage.txtBuyerLandline.getAttribute("value") == null
328     || pRaiseClaimPage.txtBuyerLandline.getAttribute("value")
329     .equals("")) {
330     pRaiseClaimPage.lnkBuyerLandlineEdit.click();
331     pRaiseClaimPage
332         .waitForElementVisible(pRaiseClaimPage.lnkBuyerLandlineEdit);
333     pRaiseClaimPage.txtBuyerLandline
334         .sendKeys(user.getBuyerPhone());
335 }
336 }
337 if (pRaiseClaimPage.lnkBuyerMobile.getAttribute("value") == null
338     || pRaiseClaimPage.lnkBuyerMobile.getAttribute("value")
339     .equals("")) {
340     pRaiseClaimPage
341         .waitForElementVisible(pRaiseClaimPage.lnkBuyerMobileEdit);
342     pRaiseClaimPage.txtBuyerMobile.clear();
343     pRaiseClaimPage.txtBuyerMobile.sendKeys(user.getBuyerPhone());
344     pRaiseClaimPage
345         .waitForElementVisible(pRaiseClaimPage.lnkBuyerLandlineEdit);
346     pRaiseClaimPage.txtBuyerLandline
347         .waitForElementVisible(pRaiseClaimPage.lnkBuyerLandlineEdit);
348     pRaiseClaimPage.txtBuyerLandline.clear();
349     pRaiseClaimPage.txtBuyerLandline.sendKeys(user.getBuyerPhone());
350     pRaiseClaimPage
351         .waitForElementVisible(pRaiseClaimPage.txtBuyerMobileEdit);
352 }
353 ThreadHelper.waitForSeconds(2);
354 pRaiseClaimPage.chooseClaimType(eClaim.getClaimTypeO());
355 }
356 // for subIssue
357 ThreadHelper.waitForSeconds(2);
358 pRaiseClaimPage.chooseSubIssue(eClaim.getSubIssueO());
359 if (eClaim.getSubIssueListO() == null) {
360     String subIssueItems[] = eClaim.getSubIssueListO().split(",");
361     for (int i = 0; i < subIssueItems.length; i++) {
362         subIssueItems[i] = SubIssue.valueOf(subIssueItems[i])
363             .getDescription();
364     }
365 }
366 pRaiseClaimPage.lblSubIssueType.click();
367 Assertion
368     .assertTrue(pRaiseClaimPage
369     .verifySubIssueDropDownList(subIssueItems),
370     "subIssue list is not expected as per the selected Issue Type");
371 }
372 }
373 // Invoice
374 if (eClaim.isInvoiceReceivedO().toLowerCase().equals("no")) {
375     Assertion.assertFalse(pRaiseClaimPage.lblInvoice.isDisplayed(),
376                           "Invoice Radio button should not be displayed");
377 }
378 }
379 }
380 }
381 }
382 }
383 }
384 }
385 }
386 }
387 }
388 }
389 }
390 }
391 }
392 }
393 }
394 }
395 }
396 }
397 }
398 }
399 }
400 }
401 }
402 }
403 }
404 }
405 }
406 }
407 }
408 }
409 }
410 }
411 }
412 }
413 }
414 }
415 }
416 }
417 }
418 }
419 }
420 }
421 }
422 }
423 }
424 }
425 }
426 }
427 }
428 }
429 }
430 }
431 }
432 }
433 }
434 }
435 }
436 }
437 }
438 }
439 }
440 }
441 }
442 }
443 }
444 }
445 }
446 }
447 }
448 }
449 }
450 }
451 }
452 }
453 }
454 }
455 }
456 }
457 }
458 }
459 }
460 }
461 }
462 }
463 }
464 }
465 }
466 }
467 }
468 }
469 }
470 }
471 }
472 }
473 }
474 }
475 }
476 }
477 }
478 }
479 }
480 }
481 }
482 }
483 }
484 }
485 }
486 }
487 }
488 }
489 }
490 }
491 }
492 }
493 }
494 }
495 }
496 }
497 }
498 }
499 }
499 }

```

```

390     Assertion.assertTrue(pRaiseClaimPage.lblInvoice.getDisplayed(),
391         "Invoice Radio button should be displayed");
392     pRaiseClaimPage.rbnYesReceivedInvoice.click();
393   } else if (egClaim.isInvoiceReceived().toLowerCase().equals("no")) {
394     Assertion.assertTrue(pRaiseClaimPage.lblInvoice.getDisplayed(),
395         "Invoice Radio button should be displayed");
396     pRaiseClaimPage.rbnNoReceivedInvoice.click();
397   } else {
398     Assertion.assertTrue(false,
399         "Wrong input passed for invoice element");
400   }
401 }
402
403 if (egClaim.getPreferredResolution() == null) {
404   pRaiseClaimPage.chooseBuyerPref(egClaim.getPreferredResolution());
405 } else {
406   egClaim.setPreferredResolution(PreferredResolution.NONE);
407 }
408
409 if (egClaim.getRefundPreference() != null) {
410   pRaiseClaimPage.selectRefundPreference(egClaim
411       .getRefundPreference());
412 }
413
414 // Image Mandatory
415 if (egClaim.isImageMandatory().equalsIgnoreCase("NA")) {
416   if (UserRole.getUserRole().equalsIgnoreCase("BUYER")) {
417     Assertion.assertTrue(
418         pRaiseClaimPage.chkMandatoryImage.getDisplayed() == false,
419         "Image Mandatory Checkbox should not be displayed");
420   } else if (UserRole.getUserRole().equalsIgnoreCase("CSR1")) {
421     Assertion.assertTrue(
422         pRaiseClaimPage.chkMandatoryImage
423             .isElementPresent() == false,
424         "Image Mandatory Checkbox should not be displayed");
425   }
426 }
427
428 } else if (egClaim.isImageMandatory().equalsIgnoreCase("yes")) {
429   Assertion.assertTrue(
430       pRaiseClaimPage.chkMandatoryImage.isEnabled() == false,
431       "Image Mandatory Checkbox should be disabled");
432 } else if (egClaim.isImageMandatory().equalsIgnoreCase("no")) {
433   Assertion.assertTrue(
434       pRaiseClaimPage.chkMandatoryImage.isEnabled() == true,
435       "Image Mandatory Checkbox should be enabled");
436 }
437
438 // Image disclaimer check box
439 if (egClaim.isImageIsChecked() == true) {
440   pRaiseClaimPage.chkMandatoryImage.click();
441 }
442
443 } else {
444   Assertion.assertTrue(false,
445       "Wrong input provided for Image Mandatory input");
446 }
447
448 // Image Upload
449 if (egClaim.isUploadImage() == true) {
450   try {
451     File file = new File(".");
452     pRaiseClaimPage.txtImageBrowse.sendKeys(file.getCanonicalPath()
453         + "\\src\\main\\resources\\Image\\hydrangeas.jpg");
454     pRaiseClaimPage.btnUploadImage.click();
455   } catch (Exception e) {
456     e.printStackTrace();
457   }
458 }
459
460 pRaiseClaimPage.btnAddInfo(egClaim.getSellerContactStatus());
461 pRaiseClaimPage.enterComments(egClaim.getBuyerRemarks());
462
463 // OTP verification
464 boolean isVerified = false;
465 if (egClaim.isAlreadyOTPVerified() == true) {
466   if (RaiseClaimTPService.isUserFieldOTP(userID, user.getBuyerPhone1()) == false)
467     String token = RaiseClaimTPService.sendOTP(userID,
468         user.getBuyerPhone1());
469
470     isVerified = RaiseClaimTPService.validateOTP(userID, token,
471         EGDBUtil.getOTP(userID), user.getBuyerPhone1());
472 }
473
474 pRaiseClaimPage.btnAddInfo(egClaim.getBuyerPhone1());
475
476 if (UserRole.getUserRole().equalsIgnoreCase("CSR1") & egClaim
477     .isCSR1NeedOTP() == false) {
478   if (isVerified == false) {
479     pRaiseClaimPage
480         .waitForElementToBeVisible(pRaiseClaimPage.lblOTPPopup);
481
482     pRaiseClaimPage.txtOTPNumber.sendKeys(EGDBUtil.getOTP(userID));
483     // pRaiseClaimPage.txtOTPNumber.sendKeys(EGDBUtil.getOTP(userID));
484     // pRaiseClaimPage.txtOTPNumber.sendKeys(EGDBUtil.getOTP("1341500456"));
485     pRaiseClaimPage.btnOTPConfirm.click();
486     return new RaiseClaimSuccessPage();
487   }
488 }
489
490 return pRaiseClaimPage.submitClaim();
491 }
492
493 public static RaiseClaimSuccessPage raiseClaimOTP(EbayEscrow escrow,
494     User user, UserRole userRole) throws Exception {
495
496   EgClaim egClaim = escrow.getEgClaim();
497   RaiseClaimHomePage pRaiseClaimHomePage = null;
498   if (UserRole.getUserRole().equalsIgnoreCase("BUYER")) {
499     pRaiseClaimHomePage = new BuyerRaiseClaimHomePage(
500         user.getLoginName(), "password");
501   }
502   // pRaiseClaimHomePage = new BuyerRaiseClaimHomePage("testgsb5",
503   // "password");
504
505   else if (UserRole.getUserRole().equalsIgnoreCase("CSR1")) {
506     pRaiseClaimHomePage = new RaiseClaimHomePage(
507         PowerShipUtil.adminGetProperty("eg_csr1"), "password");
508   }
509
510   // return raiseClaimOTP(pRaiseClaimHomePage, "371318146", egClaim,
511   // user,
512   // userRole);
513 }
514
515 /**
516  * @param raiseClaimHomePage
517  * @param egClaim
518  * @param user
519  * @param userRole
520  */
521 
```

```

456   pRaiseClaimPage
457     .waitForElementToBeVisible(pRaiseClaimPage.btnCancelDelete);
458   // Thread.sleep(5000);
459   Assertion.assertTrue(pRaiseClaimPage.btnCancelDelete
460       .getElements().get(0).isDisplayed(),
461       "Image Delete button is not displaying");
462 } catch (Exception e) {
463   e.printStackTrace();
464 }
465
466 pRaiseClaimPage.btnAddInfo(egClaim.getSellerContactStatus());
467 pRaiseClaimPage.enterComments(egClaim.getBuyerRemarks());
468
469 // OTP verification
470 boolean isVerified = false;
471 if (egClaim.isAlreadyOTPVerified() == true) {
472   if (RaiseClaimTPService.isUserFieldOTP(userID, user.getBuyerPhone1()) == false)
473     String token = RaiseClaimTPService.sendOTP(userID,
474         user.getBuyerPhone1());
475
476     isVerified = RaiseClaimTPService.validateOTP(userID, token,
477         EGDBUtil.getOTP(userID), user.getBuyerPhone1());
478 }
479
480 pRaiseClaimPage.btnAddInfo(egClaim.getBuyerPhone1());
481
482 if (UserRole.getUserRole().equalsIgnoreCase("CSR1") & egClaim
483     .isCSR1NeedOTP() == false) {
484   if (isVerified == false) {
485     pRaiseClaimPage
486         .waitForElementToBeVisible(pRaiseClaimPage.lblOTPPopup);
487
488     pRaiseClaimPage.txtOTPNumber.sendKeys(EGDBUtil.getOTP(userID));
489     // pRaiseClaimPage.txtOTPNumber.sendKeys(EGDBUtil.getOTP(userID));
490     // pRaiseClaimPage.txtOTPNumber.sendKeys(EGDBUtil.getOTP("1341500456"));
491     pRaiseClaimPage.btnOTPConfirm.click();
492     return new RaiseClaimSuccessPage();
493   }
494 }
495
496 return pRaiseClaimPage.submitClaim();
497 }
498
499 public static RaiseClaimSuccessPage raiseClaimOTP(EbayEscrow escrow,
500     User user, UserRole userRole) throws Exception {
501
502   EgClaim egClaim = escrow.getEgClaim();
503   RaiseClaimHomePage pRaiseClaimHomePage = null;
504   if (UserRole.getUserRole().equalsIgnoreCase("BUYER")) {
505     pRaiseClaimHomePage = new BuyerRaiseClaimHomePage(
506         user.getLoginName(), "password");
507   }
508   // pRaiseClaimHomePage = new BuyerRaiseClaimHomePage("testgsb5",
509   // "password");
510
511   else if (UserRole.getUserRole().equalsIgnoreCase("CSR1")) {
512     pRaiseClaimHomePage = new RaiseClaimHomePage(
513         PowerShipUtil.adminGetProperty("eg_csr1"), "password");
514   }
515
516   // return raiseClaimOTP(pRaiseClaimHomePage, "371318146", egClaim,
517   // user,
518   // userRole);
519 }
520
521 /**
522  * @param raiseClaimHomePage
523  * @param egClaim
524  * @param user
525  * @param userRole
526  */
527 
```

```

522     return raiseClaimOTP(pkraiseClaimHomePage, escrow.getPsdsPayID(),  
523         eGClaim, user, userRole);  
524 }  
525  
526 public static synchronized RaiseClaimSuccessPage raiseClaimOTP(  
527     RaiseClaimHomePage pkraiseClaimHomePage, String psdsPayID,  
528     EGClaim eGClaim, User user, UserRole userRole) throws Exception {  
529     RaiseClaimPage pRaiseClaimPage = pkraiseClaimHomePage  
530         .viewOrderDetails(psdsPayID);  
531     // add payment details  
532     String userID = user.getUserId();  
533     if (user != null) {  
534         ThreadLocalUserUtil.setThreadLocalUser(userID);  
535         if (pkraiseClaimPage.txtBuyerMobile.getAttribute("value") == null  
536             || pkraiseClaimPage.txtBuyerMobile.getAttribute("value")  
537             .equals("<>")) {  
538             pkraiseClaimPage.txtBuyerMobile.sendKeys(user.getPayPhone());  
539         }  
540         pkraiseClaimPage.lnkBuyerMobile.click();  
541         pkraiseClaimPage.waitForElementVisible(pkraiseClaimPage.txtBuyerMobile);  
542         pkraiseClaimPage.txtBuyerMobile.sendKeys(user.getPayPhone());  
543     }  
544  
545     if (pkraiseClaimPage.txtBuyerLandline.getAttribute("value") == null  
546         || pkraiseClaimPage.txtBuyerLandline.getAttribute("value")  
547         .equals("<>")) {  
548         pkraiseClaimPage.lnkBuyerLandline.click();  
549         pkraiseClaimPage.waitForElementVisible(pkraiseClaimPage.lnkBuyerLandlineEdit);  
550         pkraiseClaimPage.lnkBuyerLandlineEdit.click();  
551         pkraiseClaimPage.txtBuyerLandline.sendKeys(user.getPayPhone());  
552     }  
553     pkraiseClaimPage.txtBuyerLandline.sendKeys(user.getPayPhone());  
554 }  
555 }  
556 else if (pkraiseClaimPage.lnkBuyerMobile.getAttribute("value") == null  
557         || pkraiseClaimPage.lnkBuyerMobile.getAttribute("value")  
558         .equals("<>")) {  
559     pkraiseClaimPage.lnkBuyerMobile.click();  
560     pkraiseClaimPage.txtBuyerMobile.clear();  
561     pkraiseClaimPage.txtBuyerMobile.sendKeys(eGUtil.getRandCred  
562         .getProperty("C_HQCR_MOBILE"));  
563 }  
564 else if (pkraiseClaimPage.lnkBuyerLandline.getAttribute("value") == null  
565         || pkraiseClaimPage.lnkBuyerLandline.getAttribute("value")  
566         .equals("<>")) {  
567     pkraiseClaimPage.lnkBuyerLandline.click();  
568     pkraiseClaimPage.txtBuyerLandline.clear();  
569     pkraiseClaimPage.txtBuyerLandline.sendKeys(eGUtil.getRandCred  
570         .getProperty("C_HQCR_PHONE"));  
571 }  
572 pkraiseClaimPage.chooseClaimType(eGClaim.getClaimType());  
573 // For Subtype  
574 pkraiseClaimPage.chooseSubIssue(eGClaim.getSubIssue());  
575 if (!eGClaim.getSubIssueList() == null) {  
576     String subIssueItems[] = eGClaim.getSubIssueList().split(",");  
577     for (int i = 0; i < subIssueItems.length; i++) {  
578         subIssueOptions[i] = SubIssue.validateSubIssueItems[i]  
579             .getDescription();  
580     }  
581     pkraiseClaimPage.lblSubIssueType.click();  
582     Assertion.assertTrue(pkraiseClaimPage  
583         .verifySubIssueDropDownList(subIssueItems),  
584         "SubIssue list is not expected as per the selected Issue Type");  
585     pkraiseClaimPage.lblSubIssueType.click();  
586 }

```

```

592 if (!eGClaim.isInvoiceReceived().toLowerCase().equals("no")) {  
593     Assertion.assertFalse(pkraiseClaimPage.lblInvoice.isDisplayed(),  
594         "Invoice Radio button should not be displayed");  
595 } else {  
596     if (eGClaim.isInvoiceReceived().toLowerCase().equals("yes")) {  
597         Assertion.assertTrue(pkraiseClaimPage.lblInvoice.isDisplayed(),  
598             "Invoice Radio button should be displayed");  
599     } else if (eGClaim.isInvoiceReceived().toLowerCase().equals("no")) {  
600         Assertion.assertTrue(pkraiseClaimPage.lblInvoice.isDisplayed(),  
601             "Invoice Radio button should be displayed");  
602     } else if (eGClaim.isInvoiceReceived().toLowerCase().equals("both")) {  
603         Assertion.assertTrue(pkraiseClaimPage.lblInvoice.isDisplayed(),  
604             "Invoice Radio button should be displayed");  
605     } else {  
606         Assertion.assertFalse(pkraiseClaimPage.lblInvoice.isDisplayed(),  
607             "Wrong input passed for invoice element");  
608     }  
609 }  
610 if (eGClaim.getPreferredResolution() == null) {  
611     pkraiseClaimPage.chooseBuyerPref(eGClaim.getPreferredResolution());  
612 } else {  
613     eGClaim.setPreferredResolution(PreferredResolution.NONE);  
614 }  
615 if (eGClaim.getRefundPreference() != null) {  
616     pkraiseClaimPage.selectRefundPreference(eGClaim  
617         .getRefundPreference());  
618 }  
619 // Image Mandatory  
620 if (eGClaim.isImageMandatory().equalsIgnoreCase("NA")) {  
621     if (eGUser.getUserRole().equalsIgnoreCase("BUYER")) {  
622         Assertion.assertTrue(pkraiseClaimPage.chkMandatoryImage.isDisplayed() == false,  
623             "Image Mandatory Checkbox should not be displayed");  
624     } else if (eGUser.getUserRole().equalsIgnoreCase("SRL")) {  
625         Assertion.assertTrue(pkraiseClaimPage.chkMandatoryImage  
626             .isElementPresent() == false,  
627             "Image Mandatory Checkbox should not be displayed");  
628 } else if (eGClaim.isImageMandatory().equalsIgnoreCase("yes")) {  
629     Assertion.assertTrue(pkraiseClaimPage.chkMandatoryImage.isDisplayed() == true,  
630             "Image Mandatory Checkbox should be displayed");  
631 } else if (eGClaim.isImageMandatory().equalsIgnoreCase("no")) {  
632     Assertion.assertTrue(pkraiseClaimPage.chkMandatoryImage.isEnabled() == true,  
633             "Image Mandatory Checkbox should be enabled");  
634 }  
635 // Image disclaimer check box  
636 if (eGClaim.isImageDisclaimerChecked() == true) {  
637     pkraiseClaimPage.chkMandatoryImage.click();  
638 } else {  
639     Assertion.assertFalse(pkraiseClaimPage.chkMandatoryImage.isEnabled(),  
640         "Wrong input provided for Image Mandatory input");  
641 }  
642 // Image Upload  
643 if (eGClaim.isUploadImage() == true) {  
644

```

```

  ...
  try {
    File file = new File("C:\\");
    pRaiseClaimPage.txtImageBrowse.sendKeys(file.getCanonicalPath());
    pRaiseClaimPage.txtImageBrowse.sendKeys("C:\\src\\main\\resources\\Image\\Hydrangeas.jpg");
    pRaiseClaimPage.btnUploadImage.click();
    pRaiseClaimPage.waitForElementToBeVisible(pRaiseClaimPage.btnDelete);
    Thread.sleep(30000);
    Assertion.assertEquals(pRaiseClaimPage.btnDelete.getAttribute("value"),
        "Delete button is not displaying");
  } catch (Exception e) {
    e.printStackTrace();
  }
}
}

pRaiseClaimPage.btnAddInfo(egClaim.getSellerContactStatus());

if (userRole.getUserRole().equalsIgnoreCase("CSR")) {
  if (egClaim.isCSRNeedOTP() == true) {
    pRaiseClaimPage.rbnOTPVerifiedYes.click();
  } else {
    pRaiseClaimPage.rbnOTPVerifiedNo.click();
  }
}

pRaiseClaimPage.enterComments(egClaim.getBuyerRemarks());

// OTP verification
boolean isVerified = false;
if (egClaim.isAlreadyOTPVerified() == true) {
  if (RaiseClaimOTPService.isVerifiedOTP(userID, user.getDayPhone1()) == false) {
    String token = RaiseClaimOTPService.sendOTP(userID,
        user.getDayPhone1());
    isVerified = RaiseClaimOTPService.validateOTP(userID, token,
        EGDBUtil.getOTP(userID), user.getDayPhone1());
  }
}

pRaiseClaimPage.btnSubmit.click();
if (!userRole.getUserRole().equalsIgnoreCase("CSR") & egClaim
    .isCSRNeedOTP() == false) {
  if (isVerified == false) {
    pRaiseClaimPage.waitForElementToBeVisible(pRaiseClaimPage.lblOTPPopup);

    if (egClaim.getResendOTP() == null) {
      pRaiseClaimPage.txtBuyerMobile.sendKeys(EGDBUtil
          .getOTP(userID));
      // pRaiseClaimPage.txtOTPNumber.sendKeys("1341500456");
      Thread.sleep(5000);
      pRaiseClaimPage.btnOTPConfirm.click();
    } else {
      pRaiseClaimPage.btnOTPConfirm.click();
      Assertion.assertTrue(
          pRaiseClaimPage.lblInvalidOTP
              .getText()
              .contains("You have entered an invalid code. Please check your code or click on Resend to receive a new code"));
    }
    pRaiseClaimPage.btnOTPChangeNumber.click();
    pRaiseClaimPage.txtBuyerMobile.clear();
  }
}
}

```

```

  ...
  pRaiseClaimPage.txtBuyerMobile.clear();
  pRaiseClaimPage.txtBuyerMobile.sendKeys(egClaim
      .getResendOTP());
  pRaiseClaimPage.btnSubmit.click();

  pRaiseClaimPage.waitForElementToBeVisible(pRaiseClaimPage.lblResendOTPMsg);
  Assertion.assertEquals(pRaiseClaimPage.lblResendOTPMsg.getText(),
      "SMS code sent to your mobile again");
  pRaiseClaimPage.txtOTPNumber.sendKeys("1341500456");
  pRaiseClaimPage.btnOTPConfirm.click();
}

return new RaiseClaimSuccessPage();
}

public static RaiseClaimSuccessPage raiseClaimDB(EbayEscrow escrow,
    User user, UserRole userRole) throws Exception {
  EgClaim egClaim = escrow.getEgClaim();
  RaiseClaimHomePage pRaiseClaimHomePage = null;
  if (user != null) {
    pRaiseClaimHomePage = new BuyerRaiseClaimHomePage(
        user.getLoginName(), "password");
    // pRaiseClaimHomePage = new BuyerRaiseClaimHomePage("testlgsbi",
    // "password");
  } else {
    pRaiseClaimHomePage = new RaiseClaimHomePage(
        PowerShipUtil.admincred.getProperty("eg_csr1"), "password");
  }
  return raiseClaimDB(pRaiseClaimHomePage, "3666292434", egClaim,
      user, userRole);
}

public static synchronized RaiseClaimSuccessPage raiseClaimDB(
    RaiseClaimHomePage pRaiseClaimHomePage, String poisoPayId,
    EGClaim egClaim, User user, UserRole userRole) throws Exception {
  RaiseClaimPage pRaiseClaimPage = pRaiseClaimHomePage
      .viewOrderDetails(poisoPayId);
  // add Phone number
  if (user != null) {
    if (pRaiseClaimPage.txtBuyerMobile.getAttribute("value") == null
        || pRaiseClaimPage.txtBuyerMobile.getAttribute("value")
            .equals("")) {
      pRaiseClaimPage.txtBuyerMobileEdit.click();
      pRaiseClaimPage.waitForElementVisible(pRaiseClaimPage.txtBuyerMobile);
      pRaiseClaimPage.txtBuyerMobile.sendKeys(user.getDayPhone1());
    }
    if (pRaiseClaimPage.txtBuyerLandline.getAttribute("value") == null
        || pRaiseClaimPage.txtBuyerLandline.getAttribute("value")
            .equals("")) {
      pRaiseClaimPage.txtBuyerLandlineEdit.click();
      pRaiseClaimPage.txtBuyerLandline.sendKeys(user.getDayPhone1());
    }
  }
}

```

```

79         .waitForElementVisible(pRaiseClaimPage.lnkBuyerMobileLineEdit);
790         pRaiseClaimPage.txtBuyerMobileLine
791             .sendKeys(user.getHighPhone1());
792     }
793     } else {
794         pRaiseClaimPage.lnkBuyerMobileEdit.click();
795     }
796     pRaiseClaimPage.waitForElementVisible(pRaiseClaimPage.txtBuyerMobile);
797     pRaiseClaimPage.txtBuyerMobile.clear();
798     pRaiseClaimPage.txtBuyerMobile.sendKeys(eUtil.egCred
799         .getProperty("BUYER_MOBILE"));
800
801     pRaiseClaimPage.lnkBuyerMobileLineEdit.click();
802     pRaiseClaimPage.waitForElementVisible(pRaiseClaimPage.lnkBuyerMobileLineEdit);
803     pRaiseClaimPage.txtBuyerMobileLine
804         .clear();
805     pRaiseClaimPage.txtBuyerMobileLine.sendKeys(eUtil.egCred
806         .getProperty("BUYER_LT"));
807 }
808
809 pRaiseClaimPage.chooseClaimType(egClaim.getClaimType());
810
811 // for subIssue
812
813 pRaiseClaimPage.chooseSubIssue(egClaim.getSubIssue());
814
815 // Invoice
816
817 if (egClaim.isInvoiceReceived().toLowerCase().equals("no")) {
818     Assertion.assertFalse(pRaiseClaimPage.lblInvoice.isDisplayed(),
819         "Invoice Radio button should not be displayed");
820
821 } else if (egClaim.isInvoiceReceived().toLowerCase().equals("yes")) {
822     Assertion.assertTrue(pRaiseClaimPage.lblInvoice.isDisplayed(),
823         "Invoice Radio button should be displayed");
824     pRaiseClaimPage.rbnNotReceivedInvoice.click();
825
826 } else {
827     Assertion.assertTrue(false,
828         "Wrong input passed for invoice element");
829 }
830
831 if (egClaim.getPreferredResolution() != null) {
832     pRaiseClaimPage.chooseBuyerPref(egClaim.getPreferredResolution());
833
834 } else if (egClaim.getRefundPreference() != null) {
835     pRaiseClaimPage.selectRefundPreference(egClaim
836         .getRefundPreference());
837
838 }
839
840 // Image Mandatory
841 if (egClaim.isImageMandatory().equalsIgnoreCase("NA")) {
842     if (UserRole.getUserRole().equalsIgnoreCase("BUYER")) {
843         Assertion.assertTrue(true,
844             "Image Mandatory checkbox should be displayed");
845     }
846
847 } else if (egClaim.isImageMandatory().equalsIgnoreCase("NO")) {
848     if (UserRole.getUserRole().equalsIgnoreCase("BUYER")) {
849         Assertion.assertFalse(pRaiseClaimPage.chkMandatoryImage
850             .isElementPresent(),
851             "Image Mandatory checkbox should not be displayed");
852     }
853
854 } else if (egClaim.isImageMandatory().equalsIgnoreCase("YES")) {
855     if (UserRole.getUserRole().equalsIgnoreCase("SELLER")) {
856         Assertion.assertTrue(pRaiseClaimPage.chkMandatoryImage
857             .isEnabled(),
858             "Image Mandatory Checkbox should be enabled");
859     }
860
861 } else if (egClaim.isImageMandatory().equalsIgnoreCase("CSK1")) {
862     Assertion.assertFalse(pRaiseClaimPage.chkMandatoryImage
863         .isElementPresent(),
864             "Image Mandatory Checkbox should not be displayed");
865 }
866
867 } else if (egClaim.isImageMandatory().equalsIgnoreCase("YES")) {
868     Assertion.assertTrue(
869         pRaiseClaimPage.chkMandatoryImage.isEnabled() == false,
870         "Image Mandatory Checkbox should be disabled");
871 } else if (egClaim.isImageMandatory().equalsIgnoreCase("NO")) {
872     Assertion.assertTrue(
873         pRaiseClaimPage.chkMandatoryImage.isEnabled() == true,
874         "Image Mandatory Checkbox should be enabled");
875
876 } else if (egClaim.isImageMandatory().equalsIgnoreCase("true")) {
877     pRaiseClaimPage.chkMandatoryImage.click();
878
879 } else {
880     Assertion.assertTrue(false,
881         "Wrong input provided for Image Mandatory input");
882 }
883
884 // Image Upload
885
886 if (egClaim.isUploadImage() == true) {
887     try {
888         File file = new File("C:\\");
889         pRaiseClaimPage.lnkImageBrowse.sendKeys(file.getCanonicalPath()
890             + "\\src\\main\\resources\\Image\\Hydrangeas.jpg");
891         pRaiseClaimPage.btnUpLoadImage.click();
892
893         pRaiseClaimPage
894             .waitForElementVisible(pRaiseClaimPage.btnImgDelete);
895         ThreadHelper.waitForSeconds(30);
896         Assertion.assertTrue(pRaiseClaimPage.btnImgDelete
897             .get(0).isDisplayed(),
898             "Image Delete button is not displaying");
899     } catch (Exception e) {
900         e.printStackTrace();
901     }
902 }
903
904
905 // Beneficiary address
906 if (UserRole.getUserRole().equalsIgnoreCase("SELLER")) {
907     Assertion.assertFalse(pRaiseClaimPage.lblBenAddressSectionLabel
908         .isElementPresent(),
909         "For CSK1 beneficiary section should not be available");
910
911 } else if (UserRole.getUserRole().equalsIgnoreCase("BUYER")
912     && egClaim.getRefundPreference().equals(
913         "Demand Draft preference.DD")) {
914     Assertion.assertFalse(
915         pRaiseClaimPage.lblBenAddressSectionLabel.isDisplayed(),
916         "For CSK1 beneficiary section should not be available");
917
918 } else if (egClaim.getRefundPreference().toString().equals("DD")) {
919     Assertion.assertTrue(pRaiseClaimPage.lblBenAddressSectionLabel
920         .isElementPresent(), "For Buyer and Refund Preference as"
921         + " Demand Draft beneficiary section should be available");
922
923     String addressAsClaim = pRaiseClaimPage.lblBenAddress.getText()
924         .toUpperCase();

```

```

925
926     if (addressAsClaim.equals("NO")) {
927         Assertion.assertTrue(true,
928             "Address as claim is NO");
929     }
930
931     if (addressAsClaim.equals("YES")) {
932         Assertion.assertTrue(true,
933             "Address as claim is YES");
934     }
935
936     if (addressAsClaim.equals("DRAFT")) {
937         Assertion.assertTrue(true,
938             "Address as claim is DRAFT");
939     }
940
941     if (addressAsClaim.equals("SELLER")) {
942         Assertion.assertTrue(true,
943             "Address as claim is SELLER");
944     }
945
946     if (addressAsClaim.equals("BUYER")) {
947         Assertion.assertTrue(true,
948             "Address as claim is BUYER");
949     }
950
951     if (addressAsClaim.equals("CSK1")) {
952         Assertion.assertTrue(true,
953             "Address as claim is CSK1");
954     }
955
956     if (addressAsClaim.equals("NA")) {
957         Assertion.assertTrue(true,
958             "Address as claim is NA");
959     }
960
961     if (addressAsClaim.equals("REFUND")) {
962         Assertion.assertTrue(true,
963             "Address as claim is REFUND");
964     }
965
966     if (addressAsClaim.equals("REFUND_DRAFT")) {
967         Assertion.assertTrue(true,
968             "Address as claim is REFUND_DRAFT");
969     }
970
971     if (addressAsClaim.equals("REFUND_SELLER")) {
972         Assertion.assertTrue(true,
973             "Address as claim is REFUND_SELLER");
974     }
975
976     if (addressAsClaim.equals("REFUND_BUYER")) {
977         Assertion.assertTrue(true,
978             "Address as claim is REFUND_BUYER");
979     }
980
981     if (addressAsClaim.equals("REFUND_CSK1")) {
982         Assertion.assertTrue(true,
983             "Address as claim is REFUND_CSK1");
984     }
985
986     if (addressAsClaim.equals("REFUND_DRAFT_CSK1")) {
987         Assertion.assertTrue(true,
988             "Address as claim is REFUND_DRAFT_CSK1");
989     }
990
991     if (addressAsClaim.equals("REFUND_SELLER_CSK1")) {
992         Assertion.assertTrue(true,
993             "Address as claim is REFUND_SELLER_CSK1");
994     }
995
996     if (addressAsClaim.equals("REFUND_BUYER_CSK1")) {
997         Assertion.assertTrue(true,
998             "Address as claim is REFUND_BUYER_CSK1");
999     }
999 }

```

```

926     pRaiseClaimPage.btnAddAddress.click();
927     String addressPopup = pRaiseClaimPage.txtBenAddress1.getValue()
928         .toUpperCase();
929     +
930     + pRaiseClaimPage.txtBenAddress2.getValue().toUpperCase()
931     +
932     + pRaiseClaimPage.txtBenCity.getValue().toUpperCase()
933     +
934     + pRaiseClaimPage.listState.getSelectedText().toUpperCase()
935     +
936     + pRaiseClaimPage.txtBenPinCode.getValue().toUpperCase()
937     +
938     + pRaiseClaimPage.txtBenCountry.getValue().toUpperCase()
939     +
940     + "n" + pRaiseClaimPage.txtBenPhone.getValue();
941
942     Assertion
943         .assertEquals(addressRaiseClaim, addressPopup,
944             "Address in the Raise Claim page and address popup is not matching");
945
946     pRaiseClaimPage.btnAddAddress.clear();
947
948     pRaiseClaimPage.btnAddAddress1.sendKeys("78, Queen Street");
949     pRaiseClaimPage.btnAddAddress2.sendKeys("");
950     pRaiseClaimPage.btnAddAddress3.sendKeys("Park Avenue");
951     pRaiseClaimPage.btnAddAddress4.sendKeys("560037");
952     Thread.sleep(2000);
953     Assertion.assertFalse(pRaiseClaimPage.txtBenCity.isEnabled(),
954         "After autopopulation city should not be editable");
955     Assertion.assertFalse(pRaiseClaimPage.txtBenCountry.isEnabled(),
956         "After autopopulation State should not be editable");
957     Assertion.assertFalse(pRaiseClaimPage.txtBenCountry.isEnabled(),
958         "After autopopulation country should not be editable");
959     pRaiseClaimPage.btnAddAddress.click();
960     pRaiseClaimPage.txtBenPhone.sendKeys("9887445678");
961     pRaiseClaimPage.btnAddAddress.click();
962     Assertion
963         .assertTrue(
964             pRaiseClaimPage.lblBenDisclaimerWarning
965                 .getText()
966                 .contains(
967                     "Please check the disclaimer confirming that your Name and Address are correct before submitting"),
968             "Disclaimer should be displayed with correct text");
969     pRaiseClaimPage.chkBenAddressDisclaimer.click();
970     pRaiseClaimPage.btnAddAddress.click();
971     Thread.sleep(3000);
972
973     addressPopup = ("78, Queen Street Park Avenue BANGALORE KARNATAKA 560037 IN"
974         + "n" + "9887445678").toUpperCase();
975     addressRaiseClaim = pRaiseClaimPage.lblBenAddress.getText();
976
977     Assertion.assertEquals(addressPopup, addressRaiseClaim,
978         "Beneficiary Address should get updated");
979
980     pRaiseClaimPage.btnAddAddress.click();
981     String benName = pRaiseClaimPage.txtBenName.getText();
982     Assertion.assertFalse(pRaiseClaimPage.txtBenName.isEnabled(),
983         "Beneficiary Name field should not be editable");
984
985     pRaiseClaimPage.updateBenName(user.getLoginName(), "Gaurav",
986         "A", "Shukla");
987     pRaiseClaimPage.btnAddAddress.click();
988     Thread.sleep(3000);

```

```

992     Assertion.assertEquals(pRaiseClaimPage.txtBenName.getValue()
993         .toUpperCase(), "Gaurav A SHUKLA",
994         "Beneficiary Name should get updated");
995
996     pRaiseClaimPage.btnAdd.click();
997
998 }
999
1000 pRaiseClaimPage.btnAdd.click();
1001 pRaiseClaimPage.btnSaveComments.click();
1002 return pRaiseClaimPage.submitClaim();
1003
1004 }
1005
1006 // Added by Saranya
1007 public static RaiseClaimSuccessPage raiseClaimUploadingImage(
1008     Escrow escrow, User user) throws Exception {
1009
1010     EGClaim egClaim = escrow.getEGClaim();
1011     RaiseClaimHomePage pRaiseClaimHomePage = null;
1012     if (user != null) {
1013         pRaiseClaimHomePage = new BuyerRaiseClaimHomePage(
1014             user.getLoginName(), "password");
1015     } else {
1016         pRaiseClaimHomePage = new RaiseClaimHomePage(
1017             PowerShipUtil.adminCred.getProperty("eg_csr1"), "password");
1018     }
1019     return raiseClaimUploadingImage(pRaiseClaimHomePage,
1020         escrow.getPaisaPayID(), egClaim, user);
1021 }
1022
1023 // Added by Saranya
1024
1025 public static synchronized RaiseClaimSuccessPage raiseClaimUploadingImage(
1026     RaiseClaimHomePage pRaiseClaimHomePage, String paisaPayId,
1027     EGClaim egClaim, User user) throws Exception {
1028
1029     RaiseClaimPage pRaiseClaimPage = pRaiseClaimHomePage
1030         .viewOrderDetails(paisaPayId);
1031
1032     // add phone number
1033     if (pRaiseClaimPage.txtBuyerMobile.getAttribute("value") == null
1034         || pRaiseClaimPage.txtBuyerMobile.getAttribute("value").equals(
1035             ""))
1036     {
1037
1038         if (user == null)
1039             !user.getDayPhone1() || null && user.getDayPhone1()
1040             .trim().equalsIgnoreCase(""));
1041         pRaiseClaimPage.linkBuyerMobileEdit.click();
1042         pRaiseClaimPage.linkBuyerMobileEdit.click();
1043         pRaiseClaimPage.linkBuyerMobileEdit.click();
1044         if (user == null)
1045             pRaiseClaimPage.txtBuyerMobile.sendKeys(eUtil.egCred
1046                 .getProperty("BUYER_MOBILE"));
1047         else
1048             pRaiseClaimPage.txtBuyerMobile
1049                 .sendKeys(user.getDayPhone1());
1050
1051     }
1052
1053 }
1054
1055 pRaiseClaimPage.chooseClaimType(egClaim.getClaimType());
1056 pRaiseClaimPage.chooseBuyerPref(egClaim.getPreferredResolution());
1057 pRaiseClaimPage.selectRefundPreference(egClaim.getRefundPreference());
1058

```

```

1059     File file = new File(".*");
1060
1061     for (int i = 1; i <= 16; i++) {
1062         pRaiseSelClaimPage.txtImageDrowse.sendKeys(file.getCanonicalPath()
1063             + "\\src\\main\\resources\\Image\\Hydrogeos.jpg");
1064         pRaiseSelClaimPage.btnUploadImage.click();
1065
1066         if (i < 16) {
1067             pRaiseSelClaimPage
1068                 .waitForElementPresent(pRaiseSelClaimPage.lblFileUploaded());
1069
1070         } else {
1071             Alert alert = WebUI.driver.getWebDriver().switchTo().alert();
1072
1073             // check if alert exists
1074             // TODO find better way
1075             Assertion.assertEquals(alert.getText(),
1076                 "You can upload only 15 images",
1077                 "Accepting more than 15 images");
1078             alert.accept();
1079
1080             Thread.sleep(1000);
1081         }
1082     }
1083     pRaiseSelClaimPage.btnAddInfrageLogin.getSellerContactStatus();
1084     pRaiseSelClaimPage.enterComments(pRaiseSelClaim.getBuyerRemarks());
1085     return pRaiseSelClaimPage.submitClaim();
1086 }
1087
1088 public static RaiseSelClaimStatusError checkRaiseSelClaimStatus(
1089     EboyEscrow escrow, UserRole userRole, String updateTransactiondate,
1090     String deliveryDate) throws Exception {
1091
1092     Egclaim egclaim = escrow.getEgclaim();
1093     List<String> columnNames = new ArrayList<String>();
1094     RaiseSelClaimStatusError error = new RaiseSelClaimStatusError();
1095     Iterator<String> iterator = EboyHttpDBModule.getEscrowDetails(
1096         escrow.getEscrowID(), columnNames);
1097
1098     String[] escrowDetails = escrowDetails.next();
1099     Date transactiondate = DateUtil.parseEscrowDB(columnNameList
1100         .indexOf("CREATION_DATE"));
1101     Date lastModifiedDate = DateUtil.parseEscrowDB(columnNameList
1102         .indexOf("LAST_MODIFIED_DATE"));
1103     escrow.setStatus(new EscrowStates(Integer
1104         .valueOf(EscrowDB.COLUMNNAME_LIST.indexOf("WORKFLOW_STATE"))),
1105         null, 0);
1106
1107     if (userRole.equals(UserRole.BUYER)) {
1108
1109         if (escrow.getEscrowStates().getEscrowStateCode() == 3081
1110             || escrow.getEscrowStates().getEscrowStateCode() == 3083) {
1111             if (DateUtils.addDays(transactiondate, 30).before(
1112                 Calendar.getInstance().getTime())) {
1113                 error.setStatusCode(error.CLAIM_FAILURE);
1114                 error.setErrorMsg(
1115                     "We have encountered a problem while creating your claim. A claim cannot be raised after 30 days from the date of delivery. You can call us at 1800 209 3229 for more information.");
1116             }
1117         }
1118
1119     else if (DateUtils.addDays(transactiondate, 30).before(
1120         Calendar.getInstance().getTime())) {
1121         error.setStatusCode(error.CLAIM_FAILURE);
1122         error.setErrorMsg(
1123             "We have encountered a problem while creating your claim. Claim cannot be raised after 30 days from the date of transaction. You can call us at 1800 209 3229 for more information",
1124             null);
1125     }
1126
1127     }
1128
1129     else if (egclaim.getClaimType().equals(ClaimType.ITEM_NOT_RECEIVED)
1130         && DateUtils.addDays(transactiondate, 7).after(
1131             transactiondate)) {
1132
1133         if ((updateTransactiondate != null && Integer
1134             .valueOf(updateTransactiondate) < 0)) {
1135             error.setStatus(error.CLAIM_FAILURE);
1136             error.setErrorMsg(
1137                 "You cannot raise an item not received claim within 7 days of transaction. Please try later or call us at 1800 209 3229 for more information",
1138                 null);
1139         }
1140
1141     }
1142
1143     else if (userRole.equals(UserRole.SR)) {
1144
1145         if (egclaim.getClaimType().equals(ClaimType.ITEM_NOT_RECEIVED)
1146             && DateUtils.addDays(transactiondate, 7).after(
1147                 transactiondate)) {
1148
1149             if ((updateTransactiondate != null && Integer
1150                 .valueOf(updateTransactiondate) < 0)) {
1151                 error.setStatus(error.CLAIM_FAILURE);
1152                 error.setErrorMsg(
1153                     "You cannot raise an item not received claim within 7 days of transaction. Please try later or call us at 1800 209 3229 for more information",
1154                     null);
1155
1156         }
1157
1158         if (error.getStatus() == null)
1159             error.setStatus(error.CLAIM_SUCCESS);
1160
1161     return error;
1162
1163     }
1164
1165     public static RaiseSelClaimStatusError getErrorMessage(String messageKey,
1166         EboyTransaction es) throws Exception {
1167
1168     RaiseSelClaimStatusError error = new RaiseSelClaimStatusError();
1169
1170     if (messageKey.equals("CLAIM_SUCCESS")) {
1171         error.setStatus(error.CLAIM_SUCCESS);
1172         error.setErrorMsg(eUtil.eGrid.getPropety(messageKey), es);
1173
1174     } else if (messageKey.equals("FAILURE")) {
1175         error.setStatus(error.CLAIM_FAILURE);
1176         error.setErrorMsg(eUtil.eGrid.getPropety(messageKey), es);
1177
1178     } else {
1179         error.setStatus(error.CLAIM_FAILURE);
1180         error.setErrorMsg("Unknown error", es);
1181
1182     }
1183
1184     return error;
1185
1186     public static void validateSubmitClaimMessage(
1187         RaiseSelClaimSuccessPage raiseSelClaimSuccessPage,
1188         RaiseSelClaimStatusError error) {
1189
1190
1191     if (error.getStatus().equals(error.CLAIM_SUCCESS)) {
1192
1193         Assertion.assertEquals(raiseSelClaimSuccessPage.getSuccessMessage(),
1194             "Claim successfully registered",
1195             "Wrong claim success message is displayed");
1196         Assertion
1197             .assertEquals(
1198

```

```

1199
1200             error.getSuccessMessage(),
1201             "Wrong claim success message is displayed");
1202
1203     }
1204
1205     public static void validateSubmitClaimMessage(
1206         RaiseSelClaimSuccessPage raiseSelClaimSuccessPage,
1207         RaiseSelClaimStatusError error) {
1208
1209
1210     if (error.getStatus().equals(error.CLAIM_SUCCESS)) {
1211
1212         Assertion.assertEquals(raiseSelClaimSuccessPage.getSuccessMessage(),
1213             "Claim successfully registered",
1214             "Wrong claim success message is displayed");
1215
1216         Assertion
1217             .assertEquals(
1218

```

```

1192     assertEquals(
1193         pklSetClaimSuccessPage.getSuccessMessage1(),
1194         "Please save the claimid for future reference and communication.",
1195         "Wrong claim success message is displayed");
1196     Assertion.assertEquals(
1197         pklSetClaimSuccessPage.getSuccessMessage1(),
1198         "Please save the claimid for future reference and communication.",
1199         "Wrong claim success message is displayed");
1200 } else {
1201     Assertion.assertEquals(
1202         pklSetClaimSuccessPage.getSuccessMessage1(),
1203         "We have encountered a problem while creating your claim. Please try again later.",
1204         "Wrong error message is displayed");
1205     Assertion.assertEquals(pklSetClaimSuccessPage.getSuccessMessage1(),
1206         error.getErrorMsg(), "Wrong error message is displayed");
1207 }
1208 Assertion.assertEquals(pklSetClaimSuccessPage.getSuccessMessage1(),
1209         error.getErrorMsg(), "Wrong error message is displayed");
1210
1211 // Or call us at 1800 209 5229
1212 Assertion.assertEquals(pklSetClaimSuccessPage.getSuccessMessage1(),
1213         "Wrong error message is displayed");
1214
1215 public static void validateClaimInDB(EGClaim egClaim, UserRole role,
1216     User user) throws Exception {
1217     try {
1218         EGClaim dbEGClaim = new EGClaim(egClaim.getClaimId());
1219
1220         Assertion.assertEquals(dbEGClaim.getClaimStatus(),
1221             egClaim.getClaimStatus(), "Claim status does not match");
1222         Assertion.assertEquals(dbEGClaim.getClaimType(),
1223             egClaim.getClaimType(), "Claim type does not match");
1224         // For subtype and invoice
1225         Assertion.assertEquals(dbEGClaim.getSubIssue(),
1226             egClaim.getSubIssue(), "Sub Issue type does not match");
1227
1228         Assertion.assertEquals(dbEGClaim.isInvoiceReceived(), toUpperCaseO,
1229             egClaim.isInvoiceReceived(), toUpperCaseO,
1230             "Invoice Received Question's value does not match");
1231
1232         Assertion.assertEquals(dbEGClaim.getSellerContactStatus(),
1233             egClaim.getSellerContactStatus(),
1234             "Seller contact does not match");
1235         Assertion.assertEquals(dbEGClaim.getPreferredResolution(),
1236             egClaim.getPreferredResolution(),
1237             "Preferred resolution does not match");
1238         Assertion.assertEquals(dbEGClaim.getPaymentMethod().trim(),
1239             toUpperCaseO, egClaim.getPaymentMethod().toUpperCaseO,
1240             .trim(), "Payment method does not match");
1241     if (role.getRoleId().equals("BUYER")) {
1242         Assertion.assertEquals(
1243             dbEGClaim.isImageMandatory(), contains(
1244                 egClaim.isImageMandatory() ? "true" : "false"),
1245                 "Image mandatory value is not matching in DB");
1246     } else if (role.getRoleId().equals("SR1")) {
1247         Assertion.assertTrue(dbEGClaim.isImageMandatory()
1248             .contains("NA"),
1249             "Image mandatory value is not matching in DB");
1250     }
1251
1252     // Number DTP Verified
1253     if (role.getUserRoleO.equalsIgnoreCase("BUYER")) {
1254         Assertion.assertEquals(isClaimOTPService.isVerifiedDTPUser
1255             .getUserIdO,
1256             egClaim.getResendDTPtoO == null ? user.getBuyPhoneO
1257                 : user.getResendDTPtoO, dbEGClaim
1258                 .isAlreadyOTPVerified());
1259
1260     }
1261     // Is verified value in DB is not as expected;
1262     if (role.getUserRoleO.equalsIgnoreCase("CS1")) {
1263         Assertion.assertEquals(dbEGClaim.isCS1NeedDTP == true)
1264             .getResendDTPtoO == null ? user.getBuyPhoneO
1265                 : user.getResendDTPtoO, dbEGClaim
1266                 .isAlreadyOTPVerified());
1267     }
1268     // Is verified value in DB is not as expected;
1269     if (role.getUserRoleO.equalsIgnoreCase("CS1"))
1270         && egClaim.isCS1NeedDTP == false) {
1271         Assertion.assertEquals(false, dbEGClaim.isAlreadyOTPVerifiedO,
1272             "Is verified value in DB is not as expected");
1273
1274     }
1275     // user.getBuyPhoneO()---false{
1276     if (egClaim.getConvenienceFee() == null)
1277         Assertion.assertEquals(Double.parseDouble(dbEGClaim
1278             .getConvenienceFee().trim()), Double
1279             .parseDouble("0.00", true),
1280             "COB Convenience Fee does not match");
1281     else
1282         Assertion.assertEquals(Double.parseDouble(dbEGClaim
1283             .getConvenienceFee().trim()), Double
1284             .parseDouble(egClaim.getConvenienceFee().trim()),
1285             "COB Convenience Fee does not match");
1286
1287     if (egClaim.getSellerProtection() != null)
1288         Assertion.assertEquals(dbEGClaim.getSellerProtection(),
1289             egClaim.getSellerProtection(),
1290             "Seller Protection Status does not match");
1291     Assertion.assertEquals(dbEGClaim.getBuyerRemarksO,
1292         egClaim.getBuyerRemarksO, "remarks do not match");
1293
1294 } catch (Exception e) {
1295     throw new Exception(
1296         "Unable to validate claim details against data saved in DB");
1297 }
1298
1299 }
1300
1301 public static void validateAddressInDB(EbayTransaction es)
1302     throws Exception {
1303     try {
1304         // String hostId =
1305         // EbPayIndia08Module.getUserHostByDtor.loginName("testgsb1");
1306         String hostId = es.getBuyer().getUserHost();
1307         String addressIdO = EbUtil.getBeneficiaryAddressId(es
1308             .getUser());
1309         // User user = EbPayIndia08Module.getUserById(hostId);
1310         // User user = EbUtil.getUserById(hostId);
1311         // EbPayIndia08Module.getIpAddressByAddressId(addressIdO, "1332726024",
1312         // hostId); // (es.getBuyer().getUserID(), hostId);
1313         User user = EbUtil.getUserById(hostId);
1314         addressIdO = es.getBuyer().getUserID().toString() + hostId; // (es.getBuyer().getUserID().toString(), hostId);
1315
1316         String addressB0 = user.getAddressO + " " + user.getAddressXO
1317             + " " + user.getCityO + " " + user.getStateO + " " +
1318             + user.getZipO + " IN" + user.getBuyPhoneO;
1319         String addressB1O = user.getIdO;
1320
1321         Assertion.assertEquals(es.getEscrowListO.get(0).getEgClaimO
1322             .getBeneficiaryAddressO, addressB0,
1323             "Address should be same");
1324     } catch (Exception e) {
1325         throw new Exception(
1326             "Address should be same");
1327     }
1328 }

```

```

1295     // String hostId =
1296     // EbPayIndia08Module.getUserHostByDtor.loginName("testgsb1");
1297     String hostId = es.getBuyer().getUserHost();
1298     String addressIdO = EbUtil.getBeneficiaryAddressId(es
1299         .getUser());
1300     // User user = EbPayIndia08Module.getUserById(hostId);
1301     // EbUtil.getUserById(hostId);
1302     User user = EbUtil.getUserById(hostId);
1303     addressIdO = es.getBuyer().getUserID().toString() + hostId; // (es.getBuyer().getUserID().toString(), hostId);
1304
1305     String addressB0 = user.getAddressO + " " + user.getAddressXO
1306         + " " + user.getCityO + " " + user.getStateO + " " +
1307         + user.getZipO + " IN" + user.getBuyPhoneO;
1308     String addressB1O = user.getIdO;
1309
1310     Assertion.assertEquals(es.getEscrowListO.get(0).getEgClaimO
1311         .getBeneficiaryAddressO, addressB0,
1312         "Address should be same");
1313
1314 } catch (Exception e) {
1315     throw new Exception(
1316         "Address should be same");
1317 }

```

```

1326     }
1327     "Unable to validate claim details against data saved in DB";
1328   }
1329 
1330   public static void validateTrackingInDB(EGClaim egClaim) throws Exception {
1331     List<PSTracking> populateTracking = populateTracking(egClaim);
1332     int i = 0;
1333     for (PSTracking tracking : egClaim.getEgShipmentListO) {
1334       PSTracking dbTracking = populateTracking.get(i);
1335       try {
1336         if (tracking.getAirWayBillNo() == null)
1337           Assertion.assertEquals(dbTracking.getAirWayBillNo(),
1338             tracking.getAirWayBillNo());
1339         "Airway Bill no does not match ";
1340         if (tracking.getCarrier() == null)
1341           Assertion.assertEquals(dbTracking.getCarrier(),
1342             tracking.getCarrier());
1343         "carrier does not match ";
1344         if (tracking.getOrderType() == null)
1345           Assertion.assertEquals(dbTracking.getOrderType(),
1346             tracking.getOrderType());
1347         "order type does not match ";
1348         if (tracking.getPackageID() != null)
1349           Assertion.assertEquals(dbTracking.getPackageID(),
1350             tracking.getPackageID());
1351         "package does not match ";
1352         if (tracking.getScheduledPickupDate() != null) {
1353           dbTracking.getScheduledPickupDate().setSeconds(0);
1354           dbTracking.getScheduledPickupDate().setMinutes(0);
1355           dbTracking.getScheduledPickupDate().setHours(0);
1356           dbTracking.getScheduledPickupDate().setSeconds(0);
1357           tracking.getScheduledPickupDate().setSeconds(0);
1358           tracking.getScheduledPickupDate().setMinutes(0);
1359           tracking.getScheduledPickupDate().setHours(0);
1360           tracking.getScheduledPickupDate().setSeconds(0);
1361           tracking.getScheduledPickupDate().setMinutes(0);
1362           tracking.getScheduledPickupDate().setHours(0);
1363           Assertion.assertEquals(dbTracking.getScheduledPickupDate(),
1364             tracking.getScheduledPickupDate());
1365           "schedule pick up date does not match ";
1366         }
1367         if (tracking.getShipmentBookedDate() != null)
1368           Assertion.assertEquals(dbTracking.getShipmentBookedDateO,
1369             tracking.getShipmentBookedDateO);
1370           .setSeconds(0)
1371           + dbTracking.getShipmentBookedDateO.setSeconds(0)
1372           + dbTracking.getShipmentBookedDateO.setMinutes(0);
1373           tracking.getShipmentBookedDateO.setSeconds(0)
1374           + tracking.getShipmentBookedDateC
1375           .setSeconds(0)
1376           + tracking.getShipmentBookedDateC.setSeconds(0);
1377           "booked date does not match ";
1378         if (tracking.getActionID() != null)
1379           Assertion.assertEquals(dbTracking.getActionID(),
1380             tracking.getActionID());
1381         "action id does not match ";
1382         if (tracking.getWeightInKG() != 0)
1383           Assertion.assertEquals(dbTracking.getWeightInKG(),
1384             tracking.getWeightInKG());
1385         "weight does not match ";
1386       }
1387     }
1388   }
1389   catch (Exception e) {
1390     throw new Exception(
1391       "Unable to validate claim details against data saved in DB");
1392   }
1393 }

```

```

1395   public static List<Action> populateActionsFromDB(Integer claimID)
1396   throws Exception {
1397     // get list of action IDs involved
1398     List<Action> actionsList = null;
1399     List<String> columnNamesList = new ArrayList<String>();
1400     Iterator<String> actions = EdbUtil.getActions(claimID,
1401     columnNamesList);
1402     int tmp = 0;
1403     Action action = null;
1404     List<String> remarksList = null;
1405 
1406     while (actions.hasNext()) {
1407       actionsList = new ArrayList<Action>;
1408       String rs[] = actions.next();
1409       String rs[] = actions.next();
1410       if (tmp == 0) {
1411         // assign temp to new action id if tmp == 0 or next action
1412         // is integer value of previous action
1413         tmp = Integer.valueOf(rs[columnNamesList.indexOf("ACTION_ID")]);
1414         action = new Action();
1415         actionsList.add(action);
1416         remarksList = action.getRemarksList();
1417       }
1418       else {
1419         action.setActionID(Integer.valueOf(rs[columnNamesList
1420           .indexOf("ACTION_ID")]));
1421         action.setActionName(rs[columnNamesList
1422           .indexOf("ACTION_TYPE")]);
1423         action.setUserName(rs[columnNamesList.indexOf("CSR_USER_NAME")]);
1424         action.setPriority(rs[columnNamesList.indexOf("CSR_PRIORITY")]);
1425         action.setStatus(rs[columnNamesList.indexOf("CSR_STATUS")]);
1426         action.setActionStatus(Integer.valueOf(rs[columnNamesList
1427           .indexOf("ACTION_STATUS")]));
1428         remarksList.add(rs[columnNamesList.indexOf("CSR_REMARK")]);
1429       }
1430       tmp++;
1431     }
1432     // if some action is iterated, populate only dynamic fields
1433     else {
1434       action.setActionStatus(Integer.valueOf(rs[columnNamesList
1435           .indexOf("ACTION_STATUS")]));
1436       remarksList.add(rs[columnNamesList.indexOf("CSR_REMARK")]);
1437       if (action.getActionStatus() == 2) {
1438         tmp = 0;
1439       }
1440     }
1441   }
1442 
1443   }
1444 
1445   return actionsList;
1446 }
1447 
1448   public static void validateTracking(EGClaim claim, Error error,
1449     RaiseInboxMessage raiseInboxMessage) {
1450   Assertion.assertEquals(dbTracking.getRaiseInboxMessage(),
1451     raiseInboxMessage);
1452   "Raise Claim Error message does not match";
1453 }
1454 
1455   // write a method to get
1456 
1457   public static PSTracking getReturnTracking(EGClaim claim) {
1458     List<PSTracking> egShipmentList = claim.getEgShipmentListO;
1459     for (PSTracking shipment : egShipmentList) {
1460

```

```

1452     if (shipment.getorderType().equals("RETURN"))
1453         return shipment;
1454     }
1455     }
1456     return null;
1457 }
1458 
1459 public static String condonizeAMC() {
1460     return String.valueOf(Calendar.getInstance().getTimeInMillis());
1461 }
1462 
1463 public static List<PSTracking> populateTracking(EGoClaim claim)
1464     throws Exception {
1465     SimpleDateFormat format = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
1466     SimpleDateFormat format1 = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
1467     format.setTimeZone(TimeZone.getTimeZone("Asia/Calcutta"));
1468 
1469     PSTracking tracking = new PSTracking();
1470     List<PSTracking> shipmentList = new ArrayList<PSTracking>;
1471     List<String> columnNameList = new ArrayList<String>;
1472     for (Action a : claim.getActionList()) {
1473         if (a.getActionID() == claim.getActionID()) {
1474             Iterator<String> trackingDetails = EGUtil
1475                 .getTrackingDetails(claim.getClaimID(),
1476                     a.getActionID(), columnNameList);
1477             String[] next = trackingDetails.next();
1478             tracking.setAirWaybillNo(next[columnNameList
1479                 .indexOf("AWB NUMBER")]);
1480             tracking.setCarrierID(next[columnNameList
1481                 .indexOf("CARRIER_ID")]);
1482             tracking.setOrderType(determineShipmentType(Integer
1483                 .valueOf(next[columnNameList.indexOf("ACTION_ID")])));
1484             tracking.setTrackingNumber(next[columnNameList
1485                 .indexOf("TRACKING REF NO")]);
1486             if (next[columnNameList.indexOf("PICKUP_DATE")] != null)
1487                 tracking.setScheduledPICKUP(next[columnNameList
1488                     .indexOf("PICKUP_DATE")]);
1489             tracking.setShipmentBookedDate(next[columnNameList
1490                 .indexOf("PACKAGE_ID")]);
1491             tracking.setShipmentWeightInKg(Double.valueOf(next[columnNameList
1492                 .indexOf("PACKAGE_WEIGHT")) / 1000));
1493         }
1494         shipmentList.add(tracking);
1495     }
1496     return shipmentList;
1497 }
1498 
1499 public static boolean canShipmentBeAssociatedWithAction(int actionID) {
1500     if (Arrays.binarySearch(shipmentActionsArray, actionID) > 0)
1501         return true;
1502     else
1503         return false;
1504 }
1505 
1506 public static String determineShipmentType(int actionID) {
1507     if (Arrays.binarySearch(returnShipmentActionIDs, actionID) > 0)
1508         return "RETURN";
1509     else if (Arrays.binarySearch(replacementShipmentActionIDs, actionID) > 0)
1510         return "REPLACEMENT";
1511     else
1512         return null;
1513 }

```

```

1529     return "REPLACEMENT";
1530     else
1531         return null;
1532 }
1533 
1534 public static User getBuyerAddress(String itemId, User user,
1535     String password) throws Exception {
1536     User buyerAddress = null;
1537     try {
1538         buyerAddress = new User();
1539         GetItemTransactionsResponseType getItemTransactionsCall = Powershiputil
1540             .GetItemTransactionsCall(itemId, user.getUserId(), password);
1541         TransactionArrayType transactionArray = getItemTransactionsCall
1542             .getTransaction();
1543         TransactionType transaction = transactionArray.getTransaction()[0];
1544         AddressType shippingAddress = transaction.getBuyer().getBuyerInfo();
1545         shippingAddress.getShippingAddress();
1546 
1547         buyerAddress.setBuyerFullName(shippingAddress.getName());
1548         buyerAddress.setAddress(shippingAddress.getStreet1());
1549         buyerAddress.setCity(shippingAddress.getCityName());
1550         buyerAddress.setCountry(shippingAddress.getCountryName());
1551         buyerAddress.setZip(shippingAddress.getPostalCode());
1552         buyerAddress.setState(shippingAddress.getStateOrProvince());
1553         buyerAddress.setBuyerPhone(shippingAddress.getPhone());
1554 
1555     } catch (Exception e) {
1556         buyerAddress = EbayIndividualModule.getPrimaryShipToAddressId(
1557             user.getUserId(), null);
1558     }
1559     return buyerAddress;
1560 }
1561 
1562 public static EbayTransaction populateClaimFromDB(int claimId)
1563     throws Exception {
1564     SimpleDateFormat dbDateFormat = new SimpleDateFormat(
1565         "yyyy-MM-dd HH:mm:ss");
1566     SimpleDateFormat dbDateFormat1 = new SimpleDateFormat(
1567         "yyyy-MM-dd HH:mm:ss");
1568     SimpleDateFormat dbDateFormat2 = new SimpleDateFormat(
1569         "yyyy-MM-dd HH:mm:ss");
1570     dbDateFormat.setZone(TimeZone.getTimeZone("IST"));
1571     dbDateFormat1.setZone(TimeZone.getTimeZone("IST"));
1572     dbDateFormat2.setZone(TimeZone.getTimeZone("IST"));
1573 
1574     EbayTransaction es = new EbayTransaction();
1575     EbayEscrow escrow = new EbayEscrow();
1576     List<String> columnList = new ArrayList<String>;
1577     Iterator<String> claimDetails = EGUtil.getClaimDetails(claimId);
1578     columnNameList = claimDetails.next();
1579 
1580     String[] next = claimDetails.next();
1581 
1582     // Item details
1583     ItemType item = new ItemType();
1584     item.setItemID(next[columnNameList.indexOf("ITEM_ID")]);
1585     item.setTitle(next[columnNameList.indexOf("TITLE")]);
1586     es.setPaisaPayID(next[columnNameList.indexOf("PAISAPAY_ID")]);
1587     es.setTransactionDate(new SimpleDateFormat("yyyy-MM-dd").parse(dbDateFormat
1588         .format(next[columnNameList
1589             .indexOf("TRANSACTION_DATE")])));
1590     es.setSettlePrice(Double.parseDouble(next[columnNameList
1591             .indexOf("SETTLE_PRICE")]));
1592     escrow.setSettlePrice(Double.parseDouble(next[columnNameList
1593             .indexOf("ITEM_AMOUNT")]));
1594 }

```

```

1595     .indexof("ITEM_AMOUNT"));
1596     escrow.setItemPrice(es.getEscrowList().get(0).getItemPrice());
1597     escrow.setItemCountAmount(Double.parseDouble(next[columnNameList
1598     .indexof("ITEM_SALE_QTY"))));
1599     escrow.setConvenienceCharges(Double.parseDouble(next[columnNameList
1600     .indexof("AMOUNT_PAID"))));
1601     escrow.setConvenienceCharges(Double.parseDouble(next[columnNameList
1602     .indexof("AMOUNT_PAID"))));
1603     escrow.setPaybackAmount(Double.parseDouble(next[columnNameList
1604     .indexof("PAYBACK_AMOUNT"))));
1605     escrow.setItem(item);
1606
1607     // Claim Details
1608     EgClaim egClaim = new EgClaim();
1609     egClaim.setClaimId(next[columnNameList.indexof("claim_id")));
1610     egClaim.setClaimStatus(next[columnNameList.indexof("CLAIM_STATUS")));
1611     egClaim.setClaimCreationDate(dbDateFormat.parse(next[columnNameList
1612     .format(dbDateFormat1.parse(next[columnNameList
1613     .indexof("CREATION_TIME"))))]);
1614     egClaim.setBuyerRemarks(next[columnNameList.indexof("REMARKS")));
1615     egClaim.setBuyerRemarks(es.getEscrowList().get(0).getEgClaim());
1616     egClaim.setBuyerRemarks(es.getEscrowList().get(0).getEgClaim());
1617
1618     egClaim.setClaimType(es.getEscrowList().get(0).getEgClaim().getClaimType
1619     .valueOf(next[columnNameList.indexof("CLAIM_TYPE_ID"))));
1620
1621     egClaim.setClaimType(es.getEscrowList().get(0).getEgClaim().getClaimType
1622     .valueOf(next[columnNameList.indexof("CLAIM_TYPE_ID"))));
1623
1624     // Added by gaurav
1625
1626     egClaim.setSubIssue(subIssue.findIdByValue(
1627     .valueOf(next[columnNameList.indexof("sub_issue_id")))));
1628
1629     egClaim.setInvoiceReceived(next[columnNameList
1630     .indexof("buyer_invoice_received")));
1631
1632     egClaim.setPreferredResolution(preferredResolution.findIdByValue(
1633     .valueOf(next[columnNameList.indexof("BUYER_PREFERENCE"))));
1634
1635     egClaim.setPreferredResolution(preferredResolution.es.getEscrowList().get(0).getEgClaim());
1636
1637     egClaim.setPreferredResolution(preferredResolution);
1638
1639     egClaim.setSellerContactStatus(sellerContactStatus.findIdByValue(
1640     .valueOf(next[columnNameList.indexof("Seller_Conacted"))));
1641
1642     egClaim.setSellerContactStatus(es.getEscrowList().get(0).getEgClaim());
1643     egClaim.setSellerContactStatus(es.getEscrowList().get(0).getEgClaim());
1644
1645     egClaim.setClaimStatus(claimStatus.findIdByValue(
1646     .valueOf(next[columnNameList.indexof("CLAIM_STATUS"))));
1647     if (next[columnNameList.indexof("Seller_Protection")] != null) {
1648         egClaim.setSellerProtection(sellerProtectionStatus
1649         .findIdByValue(next[columnNameList
1650         .indexof("Seller_Protection"))));
1651     }
1652     escrow.setEgClaim(es.getEgClaim());
1653
1654     // Escrow Details
1655     escrow.setEscrowId(next[columnNameList.indexOf("escrow_id")));
1656     es.getEscrowList().add(escrow);
1657
1658     // Seller details
1659     User seller = new User();
1660     seller.setSellerName(next[columnNameList.indexOf("SELLER_NAME")));
1661     seller.setSellerID(next[columnNameList.indexOf("seller_id")));
1662
1663     // Buyer details
1664     User buyer = new User();
1665     buyer.setSellerEmail(next[columnNameList.indexOf("buyer_email")]);
1666     buyer.setSellerFirstName(next[columnNameList.indexOf("seller_first_name")));
1667     buyer.setSellerLastName(next[columnNameList.indexOf("seller_last_name")));
1668
1669     es.setSeller(seller);
1670
1671     // Buyer details
1672     buyer.setBuyerName(next[columnNameList.indexOf("BUYER_NAME")));
1673     buyer.setBuyerID(next[columnNameList.indexOf("buyer_id")));
1674     buyer.setBuyerEmail(next[columnNameList.indexOf("buyer_email_id")));
1675     buyer.setBuyerPhone(next[columnNameList.indexOf("buyer_mobile_number")));
1676     buyer.setBuyerMobile(next[columnNameList.indexOf("Buyer_Mobile_Number")));
1677
1678     buyer.setBuyerFirstName(next[columnNameList.indexOf("buyer_first_name")));
1679     buyer.setBuyerLastName(next[columnNameList.indexOf("buyer_last_name")));
1680
1681     es.setBuyer(buyer);
1682
1683     List<String> columnNamesList = new ArrayList<String>();
1684
1685     String[] shipmentDetails = null;
1686
1687     try {
1688         ShipmentDetails = EboyIndiaDBModule.getShipmentDetails(es
1689         .getEscrowList().get(0).getEscrowID(), columnNamesList);
1690     } catch (Exception e) {
1691         shipmentDetails = null;
1692     }
1693
1694     // Tracking details
1695     if (shipmentDetails != null) {
1696         SimpleDateFormat format = new SimpleDateFormat(
1697             "yyyy-MM-dd HHmmss");
1698         SimpleDateFormat format1 = new SimpleDateFormat(
1699             "yyyy-MM-dd HH:mm:ss");
1700         format.setTimeZone(TimeZone.getTimeZone("Asia/Calcutta"));
1701
1702         PSITracking tracking = new PSITracking();
1703
1704         tracking.setCarrierShippingCarrier(
1705             .findCarrierShippingCarrier(columnNamesList
1706             .indexof("carrier_shipping_carrier")));
1707         tracking.setShipmentBookedDate(dbFormat
1708             .parseShipmentDetails(columnNamesList
1709             .indexof("SHIPMENT_DATE")));
1710         tracking.setTrackingNumber(next[columnNameList
1711             .indexof("TRACKING_NUMBER")));
1712         es.getEscrowList().get(0).setTracking(tracking);
1713
1714     }
1715
1716     return es;
1717 }
1718
1719 public static boolean isTrackingNumExist(String escrowId) throws Exception {
1720     List<String> columnNamesListTracking = new ArrayList<String>();
1721     String[] trackingDetails = null;
1722     String trackingNum = null;
1723     EboyIndiaDBModule.getShipmentDetails(
1724         escrowId, columnNamesListTracking);
1725
1726     String trackingNum = trackingDetails[columnNameListTracking
1727     .indexof("TRACKING_NUMBER")];
1728
1729     if (trackingNum == null || trackingNum.equalsIgnoreCase("N/A"))
1730

```

```

1662     seller.setEmail(next[columnNameList.indexOf("seller_email")]);
1663     /*
1664      * seller.setFirstName(next[columnNameList.indexOf("seller_first_name")));
1665      * ;
1666      * seller.setLastName(next[columnNameList.indexOf("seller_last_name")));
1667      */
1668
1669     es.setSeller(seller);
1670
1671     // Buyer details
1672     buyer.setBuyerName(next[columnNameList.indexOf("BUYER_NAME")));
1673     buyer.setBuyerID(next[columnNameList.indexOf("buyer_id")));
1674     buyer.setBuyerEmail(next[columnNameList.indexOf("buyer_email_id")));
1675     buyer.setBuyerPhone(next[columnNameList.indexOf("buyer_mobile_number")));
1676     buyer.setBuyerMobile(next[columnNameList.indexOf("Buyer_Mobile_Number")));
1677
1678     buyer.setBuyerFirstName(next[columnNameList.indexOf("buyer_first_name")));
1679     buyer.setBuyerLastName(next[columnNameList.indexOf("buyer_last_name")));
1680
1681     es.setBuyer(buyer);
1682
1683     List<String> columnNamesList = new ArrayList<String>();
1684
1685     String[] shipmentDetails = null;
1686
1687     try {
1688         ShipmentDetails = EboyIndiaDBModule.getShipmentDetails(es
1689         .getEscrowList().get(0).getEscrowID(), columnNamesList);
1690     } catch (Exception e) {
1691         shipmentDetails = null;
1692     }
1693
1694     // Tracking details
1695     if (shipmentDetails != null) {
1696         SimpleDateFormat format = new SimpleDateFormat(
1697             "yyyy-MM-dd HHmmss");
1698         SimpleDateFormat format1 = new SimpleDateFormat(
1699             "yyyy-MM-dd HH:mm:ss");
1700         format.setTimeZone(TimeZone.getTimeZone("Asia/Calcutta"));
1701
1702         PSITracking tracking = new PSITracking();
1703
1704         tracking.setCarrierShippingCarrier(
1705             .findCarrierShippingCarrier(columnNamesList
1706             .indexof("carrier_shipping_carrier")));
1707         tracking.setShipmentBookedDate(dbFormat
1708             .parseShipmentDetails(columnNamesList
1709             .indexof("SHIPMENT_DATE")));
1710         tracking.setTrackingNumber(next[columnNameList
1711             .indexof("TRACKING_NUMBER")));
1712         es.getEscrowList().get(0).setTracking(tracking);
1713
1714     }
1715
1716     return es;
1717 }
1718
1719 public static boolean isTrackingNumExist(String escrowId) throws Exception {
1720     List<String> columnNamesListTracking = new ArrayList<String>();
1721     String[] trackingDetails = null;
1722     String trackingNum = null;
1723     EboyIndiaDBModule.getShipmentDetails(
1724         escrowId, columnNamesListTracking);
1725
1726     String trackingNum = trackingDetails[columnNameListTracking
1727     .indexof("TRACKING_NUMBER")];
1728
1729     if (trackingNum == null || trackingNum.equalsIgnoreCase("N/A"))
1730

```

The screenshot shows an IDE interface with multiple tabs open. The active tab is `ActionStateMappingVO.java`, displaying Java code for handling file operations and claim details. The code includes methods for reading CSV files, dealing with file paths, and interacting with a database through `EbayTransaction` objects. Other tabs visible include `ActionTypeVO.java`, `ClaimServiceError.java`, `eGResultingMessage.properties`, `CheckClaimTestPlan.java`, and `eGUtil.java`. The right side of the screen features a vertical toolbar with various icons for navigating through the code.

```
1728     if (trackNum == null || trackNum.equalsIgnoreCase("N"))
1729     {
1730         return false;
1731     }
1732 }
1733 
1734 public static String[] getDownloadedReportData(String path)
1735     throws IOException {
1736     InputStream is = null;
1737     File file = new File(path);
1738     if (!file.exists())
1739     {
1740         throw new FileNotFoundException("file : " + path
1741             + " is not available\n");
1742     }
1743     is = new FileInputStream(new File(path));
1744     return CSVUtil.read(is, CSVUtil.DELIM_CHAR);
1745 }
1746 
1747 public static String[] getDownloadedReportData(File file)
1748     throws IOException {
1749     InputStream is = null;
1750     if (!file.exists())
1751     {
1752         throw new FileNotFoundException("file : "
1753             + file.getAbsolutePath() + " is not available\n");
1754     }
1755     is = new FileInputStream(file);
1756     return CSVUtil.read(is, CSVUtil.DELIM_CHAR);
1757 }
1758 
1759 public static HashMap<Integer, EbayTransaction> populateClaimDetailsFromDB(
1760     String startDate, String endDate) throws Exception {
1761     SimpleDateFormat dateFormat = new SimpleDateFormat(
1762         "yyyy-MM-dd HH:mm:ss");
1763     SimpleDateFormat dateFormat2 = new SimpleDateFormat(
1764         "yyyy-MM-dd HH:mm:ss");
1765     dateFormat.setZone(TimeZone.getTimeZone("IST"));
1766     List<String> columnNamesList = new ArrayList<String>();
1767 
1768     HashMap<Integer, EbayTransaction> claimDetailsMapFromDB = new HashMap<Integer, EbayTransaction>;
1769     Iterator<String> iterator = EbUtil.getClaimDetails(startDate,
1770     endDate, columnNamesList);
1771     while (claimDetailsMapFromDB.hasNext()) {
1772         String[] claimDetails = claimDetailsMapFromDB.next();
1773         EbayTransaction es = new EbayTransaction();
1774         Ebyscrown escrow = new Ebyscrown();
1775         EgClaim egClaim = new EGClaim();
1776         EGClaimsParams adhocParams = new EGClaimsParams();
1777 
1778         try {
1779             int claimId = Integer.parseInt(claimDetails[columnNamesList
1780                 .indexOf("CLAIM_REQ_NO")]);
1781             es.setClaimId(claimId);
1782             escrow.setTransactionIDLong(
1783                 parseLong(claimDetails[columnNamesList
1784                     .indexOf("TRANSACTION_ID")]));
1785             escrow.setItemID(claimDetails[columnNamesList
1786                 .indexOf("ITEM_ID")]);
1787             es.setPaisaValue(claimDetails[columnNamesList
1788                 .indexOf("PAISAVALE")]);
1789             escrow.getItemID(claimDetails[columnNamesList
1790                 .indexOf("ITEM_TITLE")]);
1791             es.setTransactionDate(parseDateFormat(
1792                 format("dd/MM/yyyy", claimDetails[columnNamesList
1793                     .indexOf("TRANSACTION_DATE")))));
1794             es.setPaymentMode(claimDetails[columnNamesList
1795                     .indexOf("PAYMENT_METHOD")]);
1796             escrow.setSeller(new User());
1797         
```

The screenshot shows an IDE interface with multiple tabs open. The active tab is `ActionStateMappingVO.java`, which contains Java code for mapping claim details to a VO. The code uses `escrow` and `adhocParams` objects to set various properties based on the index of column names in a list. It handles fields like buyer name, item price, convenience charges, payback amount, saved amount, registration time, and last modified date.

```
1795     escrow.getSetter().setUserLoginName(  
1796         claimDetails[columnNameList.indexOf("SELLER_NAME")));  
1797  
1798     es.setBuyer(new User());  
1799     String buyerName = claimDetails[columnNameList.  
1800         indexOf("BUYER_NAME"));  
1801     es.getBuyerO().setUserLoginName(buyerName);  
1802  
1803     egClaim.setClaimStatus(claimStatus.findByNameId(Integer.  
1804         parseInt(claimDetails[columnNameList.  
1805             .indexOf("CLAIM_STATUS")))));  
1806     egClaim.setClaimType(claimType.findByNameId(Integer.  
1807         .valueOf(claimDetails[columnNameList.  
1808             .indexOf("CLAIM_TYPE_ID"))));  
1809     egClaim.setBuyerRemarks(claimDetails[columnNameList.  
1810         .indexOf("REMARKS")));  
1811  
1812     String itemPrice = claimDetails[columnNameList  
1813         .indexOf("ITEM_AMOUNT"));  
1814     escrow.setItemPrice(Double.parseDouble(itemPrice));  
1815     egClaim.setItemPrice(itemPrice);  
1816  
1817     escrow.setConvenienceCharges(Double.  
1818         .parseDouble(claimDetails[columnNameList  
1819             .indexOf("ITEM_CONVENIENCE_FEE"))));  
1820     escrow.setPaybackAmount(Double.  
1821         .parseDouble(claimDetails[columnNameList  
1822             .indexOf("PAYBACK_AMOUNT"))));  
1823  
1824     escrow.setIsDiscountAmount(Double.  
1825         .parseDouble(claimDetails[columnNameList  
1826             .indexOf("ITEM_SAVED"))));  
1827     adhocParams.setListingFormat(claimDetails[columnNameList  
1828         .indexOf("LISTING_FORMAT")));  
1829  
1830     es.getBuyerO().setRegistrationTime(  
1831         String.valueOf((claimDetails[columnNameList  
1832             .indexOf("BUYER_SITE"))));  
1833     escrow.setPaidAmount(Double.  
1834         .parseDouble(claimDetails[columnNameList  
1835             .indexOf("AMOUNT_PAID"))));  
1836     // egClaim.setClaimCreationTime(dbDateFormat2.parse(dbDateFormat2.format(dbDateFormat2.parse(claimDetails[columnNameList.indexOf("CREATION_TIME"))))));  
1837     // egClaim.setClaimLastModifiedTime(dbDateFormat2.parse(dbDateFormat2.format(dbDateFormat2.parse(claimDetails[columnNameList.indexOf("LAST_MODIFIED_DATE"))))));  
1838     egClaim.setClaimCreationTime(  
1839         parseInt(claimDetails[columnNameList  
1840             .indexOf("CREATION_TIME"))));  
1841     egClaim.setClaimLastModifiedDate(dbDateFormat2.  
1842         .parse(claimDetails[columnNameList  
1843             .indexOf("LAST_MODIFIED_DATE"))));  
1844  
1845     egClaim.setPreferredResolution(PreferredResolution.  
1846         findByNameId(Integer.  
1847             parseInt(claimDetails[columnNameList  
1848                 .indexOf("BUYER_PREFERENCE"))));  
1849     escrow.setSellerName(claimDetails[columnNameList  
1850         .indexOf("SELLER_NAME")));  
1851     adhocParams.setLnxType(String.  
1852         .valueOf(claimDetails[columnNameList  
1853             .indexOf("TYPE"))));  
1854     adhocParams.setVariationString(claimDetails[columnNameList  
1855             .valueOf(claimDetails[columnNameList  
1856                 .indexOf("VARIATION_ID"))));  
1857  
1858     es.getBuyerO().setBuyerPhoneC(  
1859         claimDetails[columnNameList  
1860             .indexOf("Buyer_Landline_Number")));  
1861     es.getBuyerO().setNightPhoneC
```

```

1982     claimDetails[columnNameList
1983         .indexOf("Buyer_Mobile_Number")];
1984     es.getBuyerO.setMobile(
1985         claimDetails[columnNameList.indexOf("buyer_email_id")]);
1986 
1987     escrow.setSellerO
1988         .setEmail(
1989             claimDetails[columnNameList
1990                 .indexOf("Seller_email_id")];
1991             egClaim.setSellerContactStatus(sellerContactStatus
1992                 .findbyId(Integer.parseInt(claimDetails[columnNameList
1993                     .indexOf("Seller_Contacted")]))));
1994 
1995     // adhocParams.setSellerProtection(claimDetails[columnNameList.indexOf("Seller_Protection")]);
1996     adhocParams.setSellerProtection(((claimDetails[columnNameList
1997         .indexOf("Seller_Protection")]) == null) ? "null"
1998         : ((claimDetails[columnNameList
1999             .indexOf("Seller_Protection")]));
1999 
2000     es.getBuyerO
2001         .setUserFullName(
2002             claimDetails[columnNameList
2003                 .indexOf("buyer_first_name")]
2004                 +
2005                     claimDetails[columnNameList
2006                         .indexOf("buyer_last_name")]);
2007 
2008     es.getBuyerO.setFirstName(es.getBuyerO.getUserFullName());
2009     es.getBuyerO.setLastName(es.getBuyerO.getUserFullName());
2010 
2011     escrow.getSellerO.setUserFullName(
2012         claimDetails[columnNameList
2013             .indexOf("seller_first_name")]
2014             +
2015                 claimDetails[columnNameList
2016                     .indexOf("seller_last_name")));
2017 
2018     escrow.getSellerO.setFirstName(
2019         escrow.getSellerO.getUserFullName());
2020     escrow.getSellerO.setLastName(
2021         escrow.getSellerO.getUserFullName());
2022 
2023     String createdBy = EGOUtil.getClaimCreatedBy(claimId + "");
2024 
2025     if (createdBy.equalsIgnoreCase("BUYER")) {
2026         egClaim.setClaimCreatedBy(buyerName);
2027         adhocParams.setIsSC("NO");
2028     } else {
2029         adhocParams.setIsSC("YES");
2030         egClaim.setClaimCreatedBy(createdBy);
2031     }
2032 
2033     adhocParams.setLkp((claimDetails[columnNameList
2034         .indexOf("IS_A55")];
2035     String lcs = EGOUtil.getSDetails(claimId + "");
2036     adhocParams.setIsCS(lcs));
2037 
2038     egClaim.setSmrResolved(Integer
2039         .parseInt(claimDetails[columnNameList
2040             .indexOf("SMR_RESOLVED")));
2041     egClaim.setStage(Integer.parseInt(claimDetails[columnNameList
2042         .parseInt(claimDetails[columnNameList
2043             .indexOf("CLAIM_STAGE"))]));
2044 
2045     if (!claimDetails[columnNameList.indexOf("resolution_summary")]
2046         .isEmpty())
2047         egClaim.setResolutionSummary(resolutionSummary
2048             .findnumById(Integer
2049                 .parseInt(claimDetails[columnNameList
2050                     .indexOf("resolution_summary")))));
2051 }

```

```

1930     egClaim.setEgCsvParams(adhocParams);
1931 
1932     } catch (Exception e) {
1933         e.printStackTrace();
1934     }
1935 
1936     escrow.setClaim(egClaim);
1937     List<EbayEscrow> escrowList = new ArrayList<EbayEscrow>();
1938     escrowList.add(escrow);
1939     es.setEscrowList(escrowList);
1940     claimDetailsMapFromDB.put(es.getEscrowList().get(0).getEgClaim()
1941         .getClaimId(), es);
1942 
1943     return claimDetailsMapFromDB;
1944 }
1945 
1946 public static HashMap<String, EbayTransaction> populatePSShipmentFromDB(
1947     String startDate, String endDate) throws Exception {
1948     SimpleDateFormat dbDateFormat1 = new SimpleDateFormat(
1949         "yyyy-MM-dd HH:mm:ss");
1950     dbDateFormat1.setTimeZone(TimeZone.getTimeZone("IST"));
1951     List<String> psShipmentDetailList = new ArrayList<String>();
1952     HashMap<String, psShipmentDetail> psShipmentDetailMapDB = new HashMap<String, EbayTransaction>;
1953 
1954     EbayTransaction ebayTransaction;
1955     psShipmentDetail psShipmentDetail = EGOUtil.getPSShipmentDetails(
1956         startDate, endDate, columnNameList);
1957     Iterator<String> psShipmentDetailDB = psShipmentDetailMapDB.next();
1958     while (psShipmentDetailDB.hasNext()) {
1959         String[] psShipmentDetailDBArr = psShipmentDetailDB.next();
1960         ebayTransaction = new EbayTransaction();
1961         EbayEscrow escrow = new EbayEscrow();
1962         EGClaim egClaim = new EGClaim();
1963         try {
1964             egClaim.setClaimId(Integer
1965                 .parseInt(psShipmentDetailDBArr[columnNameList
1966                     .indexOf("CLAIM_ID")]));
1967             es.setPsShipmentID(psShipmentDetailDBArr[columnNameList
1968                 .indexOf("PAISAPAY_ID")]);
1969             escrow.setAwbNumber(psShipmentDetailDBArr[columnNameList
1970                 .indexOf("AWB_NUMBER")]);
1971             // psShipmentCSV[csvRowCnt][2] - shipment status
1972             // psShipmentCSV[csvRowCnt][3] - shipment status date
1973             // psShipmentCSV[csvRowCnt][4] - booked date
1974             // psShipmentCSV[csvRowCnt][5] - pickup date
1975         } catch (Exception e) {
1976             e.printStackTrace();
1977         }
1978         es.setEgClaim(egClaim);
1979         List<EbayEscrow> escrowList = new ArrayList<EbayEscrow>();
1980         escrowList.add(escrow);
1981         es.setEscrowList(escrowList);
1982         psShipmentDetailMapDB.put(escrow.getAwbNumber(), es);
1983     }
1984 
1985     return psShipmentDetailsMapDB;
1986 }
1987 
1988 public static HashMap<Integer, EbayTransaction> populateSellerProtectionDetailsFromDB(
1989     String startDate, String endDate) throws Exception {
1990     SimpleDateFormat dbDateFormat2 = new SimpleDateFormat(
1991         "yyyy-MM-dd HH:mm:ss");
1992     SimpleDateFormat dbDateFormat1 = new SimpleDateFormat(
1993         "yyyy-MM-dd HH:mm:ss");
1994 
1995     SimpleDateFormat dbDateFormat = new SimpleDateFormat(

```

```
1996     SimpleDateFormat dbDateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
1997     dbDateFormat.setTimeZone(ZoneId.get("IST"));
1998     List<String> columnNameList = new ArrayList<String>();
1999
2000     HashMap<Integer, EbayTransaction> pShipmentDetailsMapDB = new HashMap<Integer, EbayTransaction>();
2001     // Iterator<String> sellerProtectionDetailIter = EGOUtil1.listIterator();
2002     // EGOUtil1.listIterator(sellerProtectionDetailIter);
2003     // endDate = "10:00:00", columnNameList);
2004     // endDate="10:00:00", columnNameList);
2005     Iterator<String> sellerProtectionDetailIterDB = EGOUtil1.listIterator();
2006     while (sellerProtectionDetailIterDB.hasNext()) {
2007         String[] sellerProtectionDetails = sellerProtectionDetailIterDB.next();
2008         EbayTransaction es = new EbayTransaction();
2009         EbayEscrow escrow = new EbayEscrow();
2010         EGClaim egClaim = new EGClaim();
2011         Action action = new Action();
2012         List<Action> actionList = new ArrayList<Action>();
2013
2014         EGOUtil1 egUtilDB = new EGOUtil1();
2015
2016         try {
2017             es.setTransactionDate(dbDateFormat.parse(sellerProtectionDetails[columnNameList.indexOf("TRANSACTION_DATE"))));
2018
2019             String claimID = sellerProtectionDetails[columnNameList.indexOf("CLAIM_REQ_NO"));
2020             egClaim.setClaimId(Integer.parseInt(claimID));
2021
2022             action.setActionDesc("SH100");
2023             actionList.add(action);
2024             egClaim.setActionsList(actionList);
2025
2026             action.setActionDesc("SH100");
2027             actionList.add(action);
2028             egClaim.setActionsList(actionList);
2029
2030             es.setBuyer(new User());
2031             es.getUserO().setUserLoginName(
2032                 sellerProtectionDetails[columnNameList.indexOf("BUYER_NAME")));
2033             escrow.setSeller(new User());
2034             escrow.getUserO().setUserLoginName(
2035                 sellerProtectionDetails[columnNameList.indexOf("SELLER_NAME")));
2036
2037             egClaim.setClaimId(es.getClaimId());
2038             egClaim.setClaimCreationDate(dbDateFormat.parse(sellerProtectionDetails[columnNameList.indexOf("CLAIM_STATUS"))));
2039             es.setPaisaPayID(sellerProtectionDetails[columnNameList.indexOf("PAISAPAY_ID")));
2040
2041             Double itemAmount = Double.parseDouble(sellerProtectionDetails[columnNameList.indexOf("ITEM_AMOUNT")));
2042
2043             es.setClaimAmount(itemAmount);
2044
2045             egClaim.setClaimCreationDate(dbDateFormat.parse(sellerProtectionDetails[columnNameList.indexOf("CREATION_TIME"))));
2046             egClaim.setClaimClosingDate(dbDateFormat.parse(sellerProtectionDetails[columnNameList.indexOf("CLM_CLOSED_DATE"))));
2047             egClaim.setSellerProtectionReopenedDate(dbDateFormat.parse(sellerProtectionDetails[columnNameList.indexOf("SP_CLM_OPEN_DATE"))));
2048             egClaim.setPaymentMethod(sellerProtectionDetails[columnNameList.indexOf("PAYMENT_METHOD")));
2049
2050         } catch (Exception e) {
2051             e.printStackTrace();
2052         }
2053
2054         escrow.setEgClaim(egClaim);
2055         List<EbayEscrow> escrowList = new ArrayList<EbayEscrow>();
2056         escrowList.add(escrow);
2057         es.setEscrowList(escrowList);
2058         pShipmentDetailsMapDB.put(es.getEscrowList().get(0).getEgClaimO().getClaimId(), es);
2059
2060     }
2061
2062     return pShipmentDetailsMapDB;
2063 }
2064
2065 public static HashMap<Integer, EbayTransaction> populateCODPaymentDetailsFromDB(
2066     String startDate, String endDate) throws Exception {
2067     SimpleDateFormat dbDateFormat2 = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
2068
2069     SimpleDateFormat dbDateFormat1 = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
2070     dbDateFormat1.setTimeZone(ZoneId.get("IST"));
2071     List<String> columnNameList = new ArrayList<String>();
2072
2073     HashMap<Integer, EbayTransaction> pShipmentDetailsMapDB = new HashMap<Integer, EbayTransaction>();
2074
2075     Iterator<String> codReportDetailsIterDB = EGOUtil1.getReportDetails(
2076         startDate, endDate, columnNameList);
2077
2078     while (codReportDetailsIterDB.hasNext()) {
2079         String[] codDetails = codReportDetailsIterDB.next();
2080
2081         EbayTransaction es = new EbayTransaction();
2082         EbayEscrow escrow = new EbayEscrow();
2083         EGClaim egClaim = new EGClaim();
2084         Action action = new Action();
2085         List<Action> actionList = new ArrayList<Action>();
2086
2087         EGOUtil1 egUtilDB = new EGOUtil1();
2088
2089         try {
2090             egClaim.setClaimCreationDate(dbDateFormat2.parse(codDetails[columnNameList.indexOf("CREATION_TIME"))));
2091             egClaim.setClaimClosingDate(dbDateFormat2.parse(codDetails[columnNameList.indexOf("CLM_CLOSED_DATE"))));
2092             egClaim.setClaimId(Integer.parseInt(codDetails[columnNameList.indexOf("CLAIM_REQ_NO"))));
2093             es.setPaisaPayID(codDetails[columnNameList.indexOf("PAISAPAY_ID")));
2094             es.setBuyer(new User());
2095
2096         } catch (Exception e) {
2097             e.printStackTrace();
2098         }
2099
2100         escrow.setEgClaim(egClaim);
2101         List<EbayEscrow> escrowList = new ArrayList<EbayEscrow>();
2102         escrowList.add(escrow);
2103         es.setEscrowList(escrowList);
2104         pShipmentDetailsMapDB.put(es.getEscrowList().get(0).getEgClaimO().getClaimId(), es);
2105
2106     }
2107
2108     return pShipmentDetailsMapDB;
2109 }
```

```
2064     String codee = sellerProtectionDetails[columnNameList.indexOf("CONVENIENCE_FEE"));
2065     Double codeeValue = 0.0;
2066     if (codee != null && !codee.equalsIgnoreCase("0")) {
2067         codeeValue = Double.parseDouble(codee);
2068     }
2069
2070     egClaim.setConvenienceFee(codee);
2071     Double orderValueFee = itemAmount - codeeValue;
2072     egClaim.setOrderValueFee(orderValueFee + "");
2073
2074 } catch (Exception e) {
2075     e.printStackTrace();
2076 }
2077
2078 escrow.setEgClaim(egClaim);
2079 List<EbayEscrow> escrowList = new ArrayList<EbayEscrow>();
2080 escrowList.add(escrow);
2081 es.setEscrowList(escrowList);
2082 pShipmentDetailsMapDB.put(es.getEscrowList().get(0).getEgClaimO().getClaimId(), es);
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2790
2791
2792
2793
2794
2795
2796
2797
2798
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2860
2861
2862
2863
2864
2865
2866
2867
2868
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2879
2880
2881
2882
2883
2884
2885
2886
2887
2888
2889
2890
2891
2892
2893
2894
2895
2896
2897
2898
2899
2900
2901
2902
```

```

2130     es.setBuyer(new UserO());
2131     es.setBuyerO.setID(
2132         codetools[columnNameList.indexOf("BUYER_NAME")]);
2133 
2134     escrow.setItem(new ItemTypeO());
2135     escrow.getItemO.setItemID(
2136         codetools[columnNameList.indexOf("ITEM_ID")]);
2137     escrow.getItemO.setTitle(
2138         codetools[columnNameList.indexOf("ITEM_TITLE")]);
2139 
2140     // Buyer registration details
2141     User buyerAddress = EbayFindIdBModule.getRegistrationAddress(
2142         codetools[columnNameList.indexOf("buyer_id")], null);
2143 
2144     es.getBuyerO.setBuyerAddress(es.getBuyerAddress());
2145     es.getBuyerO.setCity(es.getBuyerAddress.getCity());
2146     es.getBuyerO.setState(es.getBuyerAddress.getState());
2147     es.getBuyerO.setZip(es.getBuyerAddress.getZip());
2148 
2149     PhoneNumberType buyerPhoneType = es.getBuyerAddress.getPhoneType();
2150     phoneNumber.setPhoneNumberPart1(es.getBuyerAddress.getBuyerPhone());
2151     es.getBuyerO.setPrimaryPhone(phoneNumber);
2152 
2153     egClaim.setAmountPaid(codetools[columnNameList
2154         .indexOf("AMOUNT_PAID")]);
2155     es.getBuyerO.setUserFullName(
2156         codetools[columnNameList.indexOf("buyer_first_name")]
2157             + codetools[columnNameList
2158                 .indexOf("buyer_last_name")]);
2159 
2160     String createdBy = EBOBUtil
2161         .getClaimCreatedBy(codetools[columnNameList
2162             .indexOf("CLAIM_REQ_NO")]);
2163 
2164     if (! createdBy.equalsIgnoreCase("RECSUSER1"))
2165         createdBy = "CUSTOMER SUPPORT";
2166 
2167     egClaim.setClaimCreatedBy(createdBy);
2168 
2169     egClaim.setClaimClosedBy(EBOBUtil
2170         .getClaimCloseBy(codetools[columnNameList
2171             .indexOf("CLAIM_REQ_NO")]));
2172 
2173 
2174     es.setSeller(new UserO());
2175     es.getSellerO.setID(
2176         codetools[columnNameList.indexOf("SELLER_NAME")]);
2177     es.getSellerO.setUserID(
2178         codetools[columnNameList.indexOf("seller_id")]);
2179     egClaim.setClaimStatus(ClaimStatus.FINISHED_BY_ID(Integer
2180         .parseInt(codetools[columnNameList
2181             .indexOf("CLAIM_STATUS")).trim())));
2182 
2183     String isPaymentMethod = codetools[columnNameList
2184         .indexOf("IS_PAYMENT_REPORTED")];
2185 
2186     if (isPaymentMethod != null
2187         && Integer.parseInt(isPaymentMethod) == 4)
2188         isPaymentMethod = "YES";
2189     else
2190         isPaymentMethod = "NO";
2191 
2192     egClaim.setClaimDomainAddedAlready(isPaymentMethod);
2193 
2194 } catch (Exception e) {
2195     e.printStackTrace();
2196 }

```

```

2198     escrow.setEgClaim(egClaim);
2199     List<EbayEscrow> escrowList = new ArrayList<EbayEscrow>;
2200     ebayEscrowList.add(es);
2201     es.setEscrowList(escrowList);
2202     psShipmentDetailsMapDB.put(es.getEscrowList().get(0).getEgClaim()
2203         .getClaimID(), es);
2204 }
2205 
2206 return psShipmentDetailsMapDB;
2207 }
2208 }
2209 
2210 // Seller protection - reports
2211 public static HashMap<Integer, EbayTransaction> getRejectedClaimDetailsByCreatedDate(
2212     String sellerOracleID, String fromDate, String toDate)
2213     throws Exception {
2214     List<String> columnNames = new ArrayList<String>;
2215     Iterator<String> columnNamesI = EBOBUtil
2216         .getRejectedClaimDetailsByCreatedDate(sellerOracleID, fromDate,
2217         toDate, columnNames);
2218     return populateSellerProtectionClaimDetailsDB(claimDetailsDB,
2219         columnNamesI);
2220 }
2221 
2222 public static HashMap<Integer, EbayTransaction> getAcceptedClaimDetailsByCreatedDate(
2223     String sellerOracleID, String fromDate, String toDate)
2224     throws Exception {
2225     List<String> columnNames = new ArrayList<String>;
2226     Iterator<String> columnNamesI = EBOBUtil
2227         .getAcceptedClaimDetailsByCreatedDate(sellerOracleID, fromDate,
2228         toDate, columnNames);
2229     return populateSellerProtectionClaimDetailsDB(claimDetailsDB,
2230         columnNamesI);
2231 }
2232 
2233 public static HashMap<Integer, EbayTransaction> getAllClaimDetailsByLastUpdatedDate(
2234     String sellerOracleID, String fromDate, String toDate)
2235     throws Exception {
2236     List<String> columnNames = new ArrayList<String>;
2237     Iterator<String> columnNamesI = EBOBUtil
2238         .getAllClaimDetailsByLastUpdatedDate(sellerOracleID, fromDate,
2239         toDate, columnNames);
2240     return populateSellerProtectionClaimDetailsDB(claimDetailsDB,
2241         columnNamesI);
2242 }
2243 
2244 public static HashMap<Integer, EbayTransaction> getRejectedClaimDetailsByCreatedDate(
2245     String sellerOracleID, String fromDate, String toDate)
2246     throws Exception {
2247     List<String> columnNames = new ArrayList<String>;
2248     Iterator<String> columnNamesI = EBOBUtil
2249         .getRejectedClaimDetailsByCreatedDate(sellerOracleID, fromDate,
2250         toDate, columnNames);
2251     return populateSellerProtectionClaimDetailsDB(claimDetailsDB,
2252         columnNamesI);
2253 }
2254 
2255 public static HashMap<Integer, EbayTransaction> getAllClaimDetailsByLastUpdatedDate(
2256     String sellerOracleID, String fromDate, String toDate)
2257     throws Exception {
2258     List<String> columnNames = new ArrayList<String>;
2259     Iterator<String> columnNamesI = EBOBUtil
2260         .getAllClaimDetailsByLastUpdatedDate(sellerOracleID, fromDate,
2261         toDate, columnNames);
2262     return populateSellerProtectionClaimDetailsDB(claimDetailsDB,
2263         columnNamesI);
2264 }

```

```

2266 public static HashMap<Integer, EbayTransaction> getAcceptedClaimDetailsByLastUpdatedDate(
2267     String sellerOracleID, String fromDate, String toDate)
2268     throws Exception {
2269     List<String> columnNamesList = new ArrayList<String>();
2270     Iterator<String> claimDetailsId = EGBUtil1
2271         .getAcceptedClaimDetailsByLastUpdatedDate(sellerOracleID,
2272             fromDate, toDate, columnNamesList);
2273     return populateSellerProtectionClaimDetailsDB(claimDetailsDB,
2274         columnNamesList);
2275 }
2276
2277 public static HashMap<Integer, EbayTransaction> getRejectedClaimDetailsByLastUpdatedDate(
2278     String sellerOracleID, String fromDate, String toDate)
2279     throws Exception {
2280     List<String> columnNamesList = new ArrayList<String>();
2281     Iterator<String> claimDetailsId = EGBUtil1
2282         .getRejectedClaimDetailsByLastUpdatedDate(sellerOracleID,
2283             fromDate, toDate, columnNamesList);
2284     return populateSellerProtectionClaimDetailsDB(claimDetailsDB,
2285         columnNamesList);
2286 }
2287
2288 public static HashMap<Integer, EbayTransaction> getRejectedClaimDetailsByLastUpdatedDate(
2289     String sellerOracleID, String fromDate, String toDate)
2290     throws Exception {
2291     List<String> columnNamesList = new ArrayList<String>();
2292     Iterator<String> claimDetailsId = EGBUtil1
2293         .getRejectedClaimDetailsByLastUpdatedDate(sellerOracleID,
2294             fromDate, toDate, columnNamesList);
2295     return populateSellerProtectionClaimDetailsDB(claimDetailsDB,
2296         columnNamesList);
2297 }
2298
2299 // Check claim status reports
2300 public static HashMap<Integer, EbayTransaction> getAllCheckClaimStatusByCreatedDate(
2301     String sellerOracleID, String fromDate, String toDate)
2302     throws Exception {
2303     List<String> columnNamesList = new ArrayList<String>();
2304     Iterator<String> claimDetailsId = EGBUtil1
2305         .getAllCheckClaimStatusByCreatedDate(sellerOracleID, fromDate,
2306             toDate, columnNamesList);
2307     return populateSellerClaimDetailsDB(claimDetailsDB, columnNamesList);
2308 }
2309
2310 public static HashMap<Integer, EbayTransaction> getInProgressCheckClaimStatusByCreatedDate(
2311     String sellerOracleID, String fromDate, String toDate)
2312     throws Exception {
2313     List<String> columnNamesList = new ArrayList<String>();
2314     Iterator<String> claimDetailsId = EGBUtil1
2315         .getInProgressCheckClaimStatusByCreatedDate(sellerOracleID,
2316             fromDate, toDate, columnNamesList);
2317     return populateSellerClaimDetailsDB(claimDetailsDB, columnNamesList);
2318 }
2319
2320 public static HashMap<Integer, EbayTransaction> getClosedCheckClaimStatusByCreatedDate(
2321     String sellerOracleID, String fromDate, String toDate)
2322     throws Exception {
2323     List<String> columnNamesList = new ArrayList<String>();
2324     Iterator<String> claimDetailsId = EGBUtil1
2325         .getClosedCheckClaimStatusByCreatedDate(sellerOracleID,
2326             fromDate, toDate, columnNamesList);
2327     return populateSellerClaimDetailsDB(claimDetailsDB, columnNamesList);
2328 }
2329
2330 public static HashMap<Integer, EbayTransaction> getAllCheckClaimStatusByClosedDate(
2331     String sellerOracleID, String fromDate, String toDate)
2332     throws Exception {

```

```

2333     List<String> columnNamesList = new ArrayList<String>();
2334     Iterator<String> claimDetailsId = EGBUtil1
2335         .getAllCheckClaimStatusByClosedDate(sellerOracleID, fromDate,
2336             toDate, columnNamesList);
2337     return populateSellerClaimDetailsDB(claimDetailsDB, columnNamesList);
2338 }
2339
2340 public static HashMap<Integer, EbayTransaction> getInProgressCheckClaimStatusByClosedDate(
2341     String sellerOracleID, String fromDate, String toDate)
2342     throws Exception {
2343     List<String> columnNamesList = new ArrayList<String>();
2344     Iterator<String> claimDetailsId = EGBUtil1
2345         .getInProgressCheckClaimStatusByClosedDate(sellerOracleID,
2346             fromDate, toDate, columnNamesList);
2347     return populateSellerClaimDetailsDB(claimDetailsDB, columnNamesList);
2348 }
2349
2350 public static HashMap<Integer, EbayTransaction> getClosedCheckClaimStatusByClosedDate(
2351     String sellerOracleID, String fromDate, String toDate)
2352     throws Exception {
2353     List<String> columnNamesList = new ArrayList<String>();
2354     Iterator<String> claimDetailsId = EGBUtil1
2355         .getClosedCheckClaimStatusByClosedDate(sellerOracleID,
2356             fromDate, toDate, columnNamesList);
2357     return populateSellerClaimDetailsDB(claimDetailsDB, columnNamesList);
2358 }
2359
2360 public static HashMap<Integer, EbayTransaction> getAllCheckClaimStatusByLastUpdatedDate(
2361     String sellerOracleID, String fromDate, String toDate)
2362     throws Exception {
2363     List<String> columnNamesList = new ArrayList<String>();
2364     Iterator<String> claimDetailsId = EGBUtil1
2365         .getAllCheckClaimStatusByLastUpdatedDate(sellerOracleID,
2366             fromDate, toDate, columnNamesList);
2367     return populateSellerClaimDetailsDB(claimDetailsDB, columnNamesList);
2368 }
2369
2370 public static HashMap<Integer, EbayTransaction> getClosedCheckClaimStatusByLastUpdatedDate(
2371     String sellerOracleID, String fromDate, String toDate)
2372     throws Exception {
2373     List<String> columnNamesList = new ArrayList<String>();
2374     Iterator<String> claimDetailsId = EGBUtil1
2375         .getClosedCheckClaimStatusByLastUpdatedDate(sellerOracleID,
2376             fromDate, toDate, columnNamesList);
2377     return populateSellerClaimDetailsDB(claimDetailsDB, columnNamesList);
2378 }
2379
2380 public static HashMap<Integer, EbayTransaction> getClosedCheckClaimStatusByLastUpdatedDate(
2381     String sellerOracleID, String fromDate, String toDate)
2382     throws Exception {
2383     List<String> columnNamesList = new ArrayList<String>();
2384     Iterator<String> claimDetailsId = EGBUtil1
2385         .getClosedCheckClaimStatusByLastUpdatedDate(sellerOracleID,
2386             fromDate, toDate, columnNamesList);
2387     return populateSellerClaimDetailsDB(claimDetailsDB, columnNamesList);
2388 }
2389
2390 private static HashMap<Integer, EbayTransaction> populateSellerProtectionClaimDetailsDB(
2391     String sellerOracleID, List<String> columnNamesList)
2392     throws Exception {
2393     SimpleDateFormat dbDateFormat = new SimpleDateFormat(
2394         "yyyy-MM-dd HH:mm:ss");
2395     SimpleDateFormat dbTimeFormat = new SimpleDateFormat(
2396         "yyyy-MM-dd HH:mm:ss");
2397     dbDateFormat.setTimeZone(TimeZone.getTimeZone("IST"));
2398
2399     HashMap<Integer, EbayTransaction> claimDetailsMapFromDB = new HashMap<Integer, EbayTransaction>();
2400

```

```

2401     while (claimDetailsDB.hasNext()) {
2402         String[] claimDetails = claimDetailsDB.next();
2403         EbayTransaction es = new EbayTransaction();
2404         EbayEscrow escrow = new EbayEscrow();
2405         EgClaim egClaim = new EgClaim();
2406         EGCSVParams adhocParams = new EGCSVParams();
2407         try {
2408             egClaim.setClaimId(Integer.parseInt(claimDetails[columnNameList
2409                     .indexOf("CLAIM_REQ_NO")]));
2410             es.setTransactionIDLong(
2411                     .parseLong(claimDetails[columnNameList
2412                     .indexOf("TRANSACTION_ID")]));
2413             escrow.setItem(new ItemType());
2414             escrow.getItem().setItemID(
2415                     claimDetails[columnNameList.indexOf("ITEM_ID")]);
2416             es.setPaidByID(claimDetails[columnNameList
2417                     .indexOf("PAISAPAY_ID")]);
2418             escrow.setTitle(
2419                     claimDetails[columnNameList.indexOf("ITEM_TITLE")]);
2420
2421             String transactionDate = claimDetails[columnNameList
2422                     .indexOf("TRANSACTION_DATE")];
2423             es.setTransactionDate(dbDateFormat.parse(transactionDate));
2424             es.setPaymentMode(dbFormat2.parse(transactionDate));
2425             es.setPaymentMode(claimDetails[columnNameList
2426                     .indexOf("PAYMENT_METHOD")]);
2427
2428             escrow.setSeller(new User());
2429             escrow.setUserO().setUserName(
2430                     claimDetails[columnNameList.indexOf("SELLER_NAME")]);
2431
2432             es.setBuyer(new User());
2433             es.getBuyerO().setUserName(
2434                     claimDetails[columnNameList.indexOf("BUYER_NAME")]);
2435
2436             egClaim.setClaimStatus(claimStatus.findByNameId(Integer
2437                     .parseInt(claimDetails[columnNameList
2438                     .indexOf("CLAIM_STATUS")])));
2439             egClaim.setClaimType(ClaimType.findByNameId(Integer
2440                     .valueOf(claimDetails[columnNameList
2441                     .indexOf("CLAIM_TYPE_ID"))));
2442             egClaim.setBuyerSku(claimDetails[columnNameList
2443                     .indexOf("REMARKS")]);
2444             escrow.setUnitPrice(Double
2445                     .parseDouble(claimDetails[columnNameList
2446                     .indexOf("ITEM_AMOUNT"))));
2447             escrow.setCountAmount(Double
2448                     .parseDouble(claimDetails[columnNameList
2449                     .indexOf("AMOUNT_SAVED"))));
2450             adhocParams.setListInfo(claimDetails[columnNameList
2451                     .indexOf("VARIATION_ID")));
2452
2453             es.getBuyerO().setRegistrationSite(
2454                     String.valueOf(claimDetails[columnNameList
2455                     .indexOf("BUYER_SITE"))));
2456             escrow.setPaidAmount(Double
2457                     .parseDouble(claimDetails[columnNameList
2458                     .indexOf("AMOUNT_PAID"))));
2459             egClaim.setClaimCreationDate(dbDateFormat.parse(dbDateFormat1
2460                     .format(dbDateFormat1.parse(claimDetails[columnNameList
2461                     .indexOf("CREATE_DATE")))));
2462             egClaim.setClaimLastModified(dbDateFormat
2463                     .parse(dbDateFormat1.format(dbDateFormat
2464                     .parse(claimDetails[columnNameList
2465                     .indexOf("LAST_MODIFIED_DATE"))))));
2466             egClaim.setPreferredResolution(preferredResolution);
2467
2468         } catch (Exception e) {
2469             e.printStackTrace();
2470         }
2471
2472         // claimDetails[SVLCSRowCn[124] - IS_SS - hold
2473         // claimDetails[SVLCSRowCn[125] - Claim closed
2474
2475         egClaim.setSellerProtection(SellerProtectionStatus
2476                     .findByNameIgnoreCase(claimDetails[columnNameList
2477                     .indexOf("Seller_Protection"))));
2478         // claimDetails[SVLCSRowCn[127] - Seller comments
2479
2480         egClaim.setEGCSVParams(adhocParams);
2481
2482     } catch (Exception e) {
2483         e.printStackTrace();
2484     }
2485
2486     return claimDetailsMapFromDB;
2487 }
2488
2489 public static Date convertIST2PST(String time) throws Exception {
2490     SimpleDateFormat dateFormat = new SimpleDateFormat("dd-MM-yyyy HH:mm");
2491     dateFormat.setTimeZone(TimeZone.getTimeZone("IST"));
2492     SimpleDateFormat dateFormat1 = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
2493     dateFormat1.setTimeZone(TimeZone.getTimeZone("PST"));
2494     return dateFormat.parse(dateFormat1.format(dateFormat.parse(time)));
2495 }
2496
2497 private static Map<String[], EbayTransaction> populateSellerClaimDetailsDB(
2498     Iterator<String[]> claimDetailsDB, List<String> columnNameList)
2499     throws Exception {
2500     SimpleDateFormat dbDateFormat = new SimpleDateFormat("yyyy-MM-dd");
2501     dbDateFormat.setTimeZone(TimeZone.getTimeZone("IST"));
2502
2503     SimpleDateFormat dbDateFormat1 = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
2504     dbDateFormat1.setTimeZone(TimeZone.getTimeZone("IST"));
2505
2506     HashMap<Integer, EbayTransaction> claimDetailsMapFromDB = new HashMap<Integer, EbayTransaction>;
2507
2508     while (claimDetailsDB.hasNext()) {
2509         String[] claimDetails = claimDetailsDB.next();
2510         EbayTransaction es = new EbayTransaction();
2511         EbayEscrow escrow = new EbayEscrow();
2512         EgClaim egClaim = new EgClaim();
2513         EGCSVParams adhocParams = new EGCSVParams();
2514         try {
2515             egClaim.setClaimId(Integer.parseInt(claimDetails[columnNameList
2516                     .indexOf("CLAIM_REQ_NO")]));
2517
2518

```

```

2539     escrow.setPmsoId(0L);
2540     .parseLong(columnNameList.get(indexOf("TRANSACTION_ID")));
2541     escrow.setItem(new ItemType());
2542     escrow.getItem().setColumnIndexList(columnNameList.indexOf("ITEM_ID"));
2543     es.setItemType();
2544     es.setItemId();
2545     es.setItemDetails(columnNameList.indexOf("ITEM_ID"));
2546     es.setPaidPopUpDC(itemDetails[columnNameList.indexOf("PAISAPAY_ID")]);
2547     escrow.getItem().setLineItem(columnNameList.indexOf("ITEM_TITLE"));
2548     es.setTransactionDate(dbDateFormat3.parse(columnNameList.indexOf("TRANSACTION_DATE")));
2549     es.setPaymentMethod(itemDetails[columnNameList.indexOf("PAYMENT_METHOD")]);
2550     escrow.setSellerUser();
2551     escrow.getBuyerUser();
2552     escrow.getBuyerUser().setBuyerName();
2553     claimDetails[columnNameList.indexOf("SELLER_NAME")];
2554     es.setBuyerUser();
2555     es.getBuyerUser().setBuyerName();
2556     claimDetails[columnNameList.indexOf("BUYER_NAME")];
2557     egClaim.setClaimStatus(ClaimStatus.findByNameId(Integer.parseInt(columnNameList.indexOf("CLAIM_STATUS"))));
2558     egClaim.setClaimType(ClaimType.findByNameId(Integer.parseInt(columnNameList.indexOf("CLAIM_TYPE"))));
2559     egClaim.setBuyerRemarks(claimDetails[columnNameList.indexOf("REMARKS")]);
2560     escrow.setItemPrice(itemDetails[columnNameList.indexOf("ITEM_AMOUNT")]);
2561     escrow.setDiscountAmount(itemDetails[columnNameList.indexOf("AMOUNT_SAVED")]);
2562     adhocParams.setListingFormat(itemDetails[columnNameList.indexOf("VARIATION_ID")]);
2563     es.getBuyerUser().setRegistrationSite("String.valueOf(itemDetails[columnNameList.indexOf("BUYER_SITE")])");
2564     escrow.setPaidAmount(itemDetails[columnNameList.indexOf("AMOUNT_PAID")]);
2565     egClaim.setClaimCreateDate(dbDateFormat.parse(columnNameList.indexOf("CREATION_TIME")));
2566     egClaim.setClaimModifiedDate(dbDateFormat.parse(columnNameList.indexOf("LAST_MODIFIED_DATE")));
2567     egClaim.setPreferredResolution(PreferredResolution.findByNameId(Integer.parseInt(columnNameList.indexOf("BUYER_PREFERENCE"))));
2568     escrow.setSellerProtectionStatus(columnNameList.indexOf("SELLER_PROTECTION"));
2569     adhocParams.setBuyerType("String.valueOf(itemDetails[columnNameList.indexOf("TYPE")])");
2570     adhocParams.setVariationId("String.valueOf(itemDetails[columnNameList.indexOf("VARIATION_ID")])");
2571     adhocParams.setEgcsParams(adhocParams);

```

```

2665     egClaim.setSellerProtectionSellerProtectionStatus(
2666         .findByNameDesc(claimDetails[columnNameList.indexOf("Seller_Protection")]));
2667     // claimDetails[CSVclaimsRowCnt][24] - IS_SOLD - hold
2668     // claimDetails[CSVclaimsRowCnt][25] - Claim closed
2669     egClaim.setSellerProtectionSellerProtectionStatus(
2670         .findByNameDesc(claimDetails[columnNameList.indexOf("Seller_Protection")]));
2671     // claimDetails[CSVclaimsRowCnt][27] - Seller comments
2672     egClaim.setEgcsParams(adhocParams);
2673     } catch (Exception e) {
2674     e.printStackTrace();
2675     }
2676     escrow.setEgClaim(egClaim);
2677     List<EbayEscrow> escrowList = new ArrayList<EbayEscrow>();
2678     escrowList.add(escrow);
2679     es.setEscrowList(escrowList);
2680     claimDetailsMapFromDB.put(es.getEscrowList().get(0).getEgClaim().getClaimId(), es);
2681     }
2682     return claimDetailsMapFromDB;
2683 }
2684 public static void main(String[] args) throws Exception {
2685     * List<Action> populateActionsFromDB = populateActionsFromDB(S475);
2686     * populateActionsFromDB.toString();
2687     *
2688     */
2689     * EgClaim egClaim = new EgClaim(); egClaim.setClaimId(7832);
2690     egClaim.setClaimType(ClaimType.ITEM_NOT_AS_EXPECTED);
2691     egClaim.setPreferredResolution();
2692     (* PreferredResolution.I_WANT_A_REPLACEMENT);
2693     egClaim.setSellerProtectionStatus(TRIED_TO_SELLER_NOT_REACHABLE);
2694     egClaim.setSellerProtectionSellerProtectionStatus(open);
2695     egClaim.setClaimStatus(ClaimStatus.OPEN); validateClaimInDB(egClaim);
2696     */
2697     String date = "2013-07-21 00:00:00.0";
2698     SimpleDateFormat dbDateFormat1 = new SimpleDateFormat(
2699         "yyyy-MM-dd HH:mm:ss");
2700     SimpleDateFormat dbDateFormat4 = new SimpleDateFormat(
2701         "yyyy-MM-dd HH:mm:ss");
2702     System.out.println(dbDateFormat1.parse(date));
2703     System.out.println(dbDateFormat4.parse(date));
2704 }
2705 public static void updateTransactionStatus(EbayEscrow escrow,
2706     EbayTransaction es) throws Exception {
2707     /*
2708     * 05/17 commented as claiming changes are not displayed
2709     * this is for IHR use case
2710     */
2711     if (es.getEscrowList().get(0).getEgClaim().getClaimType() == es.getEgClaim().getClaimType()) {
2712         ebayIndiDBModule.updateEscrowINN(es.getEscrowID(),
2713             Integer.valueOf(escrowdates.getShippedDate()));
2714     }

```

```
2672     * Integer.valueOf(escrowdates.getDeliveryBeforeDate()); } if
2673     (escrowdates.getStatusCode()>= Status.DELIVERED
2674     && !tes.getEscrowList().get(0).getClaimType().equals(claimType)
2675     && tes.getEscrowList().get(0).getClaimType().equals(ClaimType.ITEM_NOT_RECEIVED)) {
2676
2677     /* SignInPage pSignInPage = new SignInPage(true);
2678     pSignInPage.signInEscrow.getSeller().getUserLoginName(),
2679     pSellerTransactionDetailsPage,
2680     pSellerTransactionDetailsPage.getDriver().quit();
2681     pSellerTransactionDetailsPage = new SellerTransactionDetailsPage(
2682     escrow.getEscrowID(), true);
2683     pSellerTransactionDetailsPage.markReadyToShip(escrow);
2684     pSellerTransactionDetailsPage.getDriver().quit();
2685
2686
2687     */
2688
2689     Calendar instance = Calendar.getInstance();
2690     instance.add(Calendar.DATE, 1); Date tempdate = instance.getTime();
2691     PowershipUtil.workReadyShip(escrow); // get the tracking details and
2692     set into escrowDetailsPage PS Tracking escrow
2693     setCombinedOrderIndicator(false);
2694     PowershipUtil.updateShipmentStatus(escrow,
2695     EbayWBStatusEnum.PICKED_IN_TRANSIT, tempdate);
2696     tempdate.setDate(tempdate.getDate() + 1);
2697     PowershipUtil.updateShipmentStatus(escrow,
2698     EbayWBStatusEnum.DELIVERED, tempdate);
2699     EbayIndiaModule.updateEscrowDeliveryDate(escrow.getEscrowID(),
2700     Integer.valueOf(escrowdates.getDeliveryDate()));
2701
2702     // /end
2703
2704     // 05/02 // this is for INR use case
2705     if (escrowdates.getCreationRange() != 0) {
2706       EbayIndiaDBModule.updateEscrowCreationDate(escrow.getEscrowID(),
2707       Integer.valueOf(escrowdates.getCreationRange()));
2708     }
2709
2710     if (escrowdates.getEscrowStatusCode() == Status.DELIVERED
2711     .getStatusCode()) {
2712
2713     /* SignInPage pSignInPage = new SignInPage(true);
2714     pSignInPage.signInEscrow.getSeller().getUserLoginName(),
2715     "password"; SellerTransactionDetailsPage
2716     pSellerTransactionDetailsPage = new SellerTransactionDetailsPage(
2717     escrow.getEscrowID(), true);
2718     pSellerTransactionDetailsPage.markReadyToShip(escrow);
2719     pSellerTransactionDetailsPage.getDriver().quit();
2720     */
2721
2722     Calendar instance = Calendar.getInstance();
2723     instance.add(Calendar.DATE, 1);
2724     Date tempdate = instance.getTime();
2725     PowershipUtil.workReadyShip(escrow);
2726     PowershipUtil.updateReadyShip(escrow);
2727
2728     // / get the tracking details and into escrow
2729     escrow.setTracking(new PSTrackingEscrow
2730     .getCombinedOrderIndicator(), false);
2731     PowershipUtil.updateShipmentStatus(escrow,
2732     EbayWBStatusEnum.PICKED_IN_TRANSIT, tempdate);
2733     tempdate.setDate(tempdate.getDate() + 1);
2734     PowershipUtil.updateShipmentStatus(escrow,
2735     EbayWBStatusEnum.DELIVERED, tempdate);
2736
2737     EbayIndiaDBModule.updateEscrowDeliveryDate(escrow.getEscrowID(),
2738
2739
2740
2741
2742     if (escrowdates.getEscrowStatusCode() == Status.SHIPPED
2743     && !tes.getEscrowList().get(0).getClaimType().equals(claimType))
2744     {
2745       SignInPage pSignInPage = new SignInPage(true);
2746       pSignInPage.signInEscrow.getSeller().getUserLoginName(),
2747       "password";
2748
2749       SellerTransactionDetailsPage pSellerTransactionDetailsPage =
2750       new SellerTransactionDetailsPage(
2751       escrow.getEscrowID(), true);
2752       pSellerTransactionDetailsPage.markReadyToShip(escrow);
2753       pSellerTransactionDetailsPage.getDriver().quit();
2754
2755       escrow.setTracking(new PSTrackingEscrow
2756       .getCombinedOrderIndicator(), false);
2757       Calendar instance = Calendar.getInstance();
2758       instance.add(Calendar.DATE, 1);
2759       Date tempdate = instance.getTime();
2760       PowershipUtil.updateShipmentStatus(escrow,
2761       EbayWBStatusEnum.PICKED_IN_TRANSIT, tempdate);
2762
2763
2764     /**
2765      * Utility function to convert java Date to TimeZone format
2766
2767      * @param date
2768      * @param format
2769      * @param timeZone
2770      * @return
2771      * @code formatDateToString(date, "dd MMM yyyy hh:mm:ss a", "IST")
2772     */
2773
2774     public static String formatDateToString(Date date, String format,
2775     String timeZone) {
2776       //null check
2777       if (date == null)
2778         return null;
2779       SimpleDateFormat sdf = new SimpleDateFormat(format);
2780       // default system timeZone if passed null or empty
2781       if (timeZone == null || "".equalsIgnoreCase(timeZone.trim())) {
2782         timeZone = Calendar.getInstance().getTimeZone().getID();
2783       }
2784       // set timeZone to SimpleDateFormat
2785       sdf.setTimeZone(TimeZone.getTimeZone(timeZone));
2786       // return date in required format with timeZone as String
2787       return sdf.format(date);
2788     }
2789
2790
2791     public static Date dateFormateInPST(String expDate, boolean isEnd)
2792     throws Exception {
2793       if (expDate == null || expDate.trim().equals(""))
2794         return null;
2795       if (isEnd) {
2796         expDate = expDate.trim() + " 23:59:59";
2797       } else {
2798         expDate = expDate.trim() + " 00:00:00";
2799       }
2800
2801       DateFormat df = new SimpleDateFormat("yyyy-MM-dd hh:mm:ss");
2802       df.setTimeZone(TimeZone.getTimeZone("IST"));
2803       Date date = df.parse(expDate);
2804
2805       return convertToPST(date);
2806     }
2807
2808     public static Date convertToPST(Date dateToConvert) {
```

```
2742     if (expDate == null || expDate.trim().equals(""))
2743       return null;
2744     if (isEnd) {
2745       expDate = expDate.trim() + " 23:59:59";
2746     } else {
2747       expDate = expDate.trim() + " 00:00:00";
2748     }
2749
2750     DateFormat df = new SimpleDateFormat("yyyy-MM-dd hh:mm:ss");
2751     df.setTimeZone(TimeZone.getTimeZone("IST"));
2752     Date date = df.parse(expDate);
2753
2754     return convertToPST(date);
2755   }
2756
2757   public static Date convertToPST(Date dateToConvert) {
```

ActionStateMappingVO.java

```
2810     TimeZone PST = timeZone.getTimeZone("PST");
2811     int dstSavings = PST.dstSavings();
2812     Calendar cl = calendar.getCalendar();
2813     cl.setTime(dateToConvert);
2814     if (dstSavings < 0 && !PST.useDaylightTime()) {
2815         cl.add(Calendar.MILLISECOND, dstSavings);
2816     }
2817     dateToConvert = cl.getTime();
2818
2819     DateFormat formatter = new SimpleDateFormat();
2820     formatter.setTimeZone(PST);
2821     Date currPST = null;
2822     try {
2823         currPST = new SimpleDateFormat().parse(formatter
2824             .format(dateToConvert));
2825     } catch (ParseException e) {
2826         // TODO Auto-generated catch block
2827         e.printStackTrace();
2828     }
2829     return currPST;
2830 }
2831 /**
2832 * Author: abhilashsingh this will convert DB PST date to IST date
2833 */
2834
2835 public static String getFormattedDISTDateTime(Date m_txndateTime) {
2836     if (m_txndateTime == null)
2837         return "";
2838     DateFormat df = new SimpleDateFormat("dd-MM-yyyy HH:mm");
2839     String reportDate = df.format(eUtil.convertToIST(m_txndateTime));
2840     return reportDate;
2841 }
2842 /**
2843 *
2844 * @param dateToConvert
2845 * @return
2846 */
2847 public static Date convertToIST(Date dateToConvert) {
2848     TimeZone IST = timeZone.getTimeZone("IST");
2849     calendar = calendar.getCalendar();
2850     calendar.setTime(dateToConvert);
2851     dateToConvert = calendar.getTime();
2852
2853     DateFormat formatter = new SimpleDateFormat();
2854     formatter.setTimeZone(IST);
2855     Date currIST = null;
2856     try {
2857         currIST = new SimpleDateFormat().parse(formatter
2858             .format(dateToConvert));
2859     } catch (ParseException e) {
2860         // TODO Auto-generated catch block
2861         e.printStackTrace();
2862     }
2863     return currIST;
2864 }
2865
2866 public static Date getDBtoCSVFormatDate(Date dbDate) {
2867     // SimpleDateFormat df=new SimpleDateFormat("dd-mm-yyyy");
2868     dbDate = eUtil.addMinutes(dbDate, 30);
2869     dbDate = DateUtils.addHours(dbDate, 12);
2870     return dbDate;
2871 }
2872
2873 public static String getDateDDMMYYYY(String inputDate, int add)
2874 }
```

ActionStateMappingVO.java

```
2877     throw new NotImplementedException();
2878     Date date = FormatSource.parse(inputDate);
2879     DateFormat formatTarget = new SimpleDateFormat("dd-mm-yyyy");
2880     Calendar c = calendar.getCalendar();
2881     c.set(Calendar.DATE, date.getDate());
2882     c.set(Calendar.MONTH, date.getMonth());
2883     c.set(Calendar.YEAR, date.getYear());
2884     return formatTarget.format(c.getTime());
2885 }
2886
2887 public static void validateNotification(UserNotification egCommunication,
2888     String claimID, String triggeredBy) throws Exception {
2889     List<String> columnNameList = new ArrayList<String>();
2890     String query = null;
2891     StringBuilder query = new StringBuilder();
2892     query.append("SELECT c.claim_req_no,c.context,c.receiver,c.triggered_by,c.comm_type,c.template_id,c.send_status,c.contact_number ")
2893     .append(" FROM eg_claim_communication c ")
2894     .append(" WHERE c.claim_req_no = ?").append(claimID)
2895     .append(" AND c.triggered_by = ?").append(triggeredBy)
2896     .append(" ORDER BY c.communication_id DESC; ");
2897     Iterator<String> result = EGBDBUtil.getResultSet(query.toString(),
2898     columnNameList);
2899
2900     while (result.hasNext()) {
2901         String[] next = result.next();
2902         String columnValue = next[columnNameList.indexOf("comm_type")];
2903         String receiver = next[columnNameList.indexOf("receiver")];
2904         if (receiver.equals("BUYER")) {
2905             if (comm_type.equals("EMAIL")) {
2906                 Logging.info("Buyer : " + next[columnNameList.indexOf("template_id")]);
2907                 Assertion.assertEquals(
2908                     next[columnNameList.indexOf("template_id")],
2909                     egCommunication.getBuyerEmailTemplateId(),
2910                     "Buyer Email template is not matching");
2911             } else {
2912                 Logging.info("Buyer : " +
2913                     next[columnNameList.indexOf("template_id")]);
2914                 Assertion.assertEquals(
2915                     next[columnNameList.indexOf("template_id")],
2916                     egCommunication.getBuyerSMSTemplateId(),
2917                     "Buyer SMS template is not matching");
2918             }
2919         } else {
2920             if (comm_type.equals("EMAIL")) {
2921                 Logging.info("Seller : " +
2922                     next[columnNameList.indexOf("template_id")]);
2923                 Assertion.assertEquals(
2924                     next[columnNameList.indexOf("template_id")],
2925                     egCommunication.getSellerEmailTemplateId(),
2926                     "Seller Email template is not matching");
2927             } else {
2928                 Logging.info("Seller : " +
2929                     next[columnNameList.indexOf("template_id")]);
2930                 Assertion.assertEquals(
2931                     next[columnNameList.indexOf("template_id")],
2932                     egCommunication.getSellerSMSTemplateId(),
2933                     "Seller SMS template is not matching");
2934             }
2935         }
2936     }
2937 }
```

```

2945 public static AntecedentSuccessPage raiseClaimNegative(EBuyEscrow escrow,
2946     User user, UserRole userRole, String resultMessage)
2947     throws Exception {
2948
2949     EGClaim egClaim = escrow.getEGClaim();
2950     RaiseClaimHomePage pRaiseClaimHomePage = null;
2951     if (user != null) {
2952         pRaiseClaimHomePage = new BuyerRaiseClaimHomePage(
2953             user.getLoginName(), "password");
2954         // pRaiseClaimHomePage = new BuyerRaiseClaimHomePage("testsgsb1",
2955         // "password");
2956     } else {
2957         pRaiseClaimHomePage = new RaiseClaimHomePage(
2958             PowerSightUtil.ADMINRED.getProperty("eg_csr1"), "password");
2959     }
2960     // return raiseClaimNew(pRaiseClaimHomePage, "3646000394", egClaim,
2961     // user, userRole);
2962     return raiseClaimNegative(pRaiseClaimHomePage, escrow.getPAisoPayID(),
2963         egClaim, user, userRole, resultMessage);
2964
2965 }
2966
2967 public static synchronized RaiseClaimSuccessPage raiseClaimNegative(
2968     RaiseClaimHomePage pRaiseClaimHomePage, String payoID,
2969     EGClaim egClaim, User user, UserRole userRole, String resultMessage)
2970     throws Exception {
2971
2972     pRaiseClaimHomePage.pRaiseClaimPage = pRaiseClaimHomePage
2973         .viewOrderDetails(pRaiseClaimPage);
2974
2975     ThreadHelper.waitForSeconds();
2976     if (pRaiseClaimPage.lblErrorMessage.isElementPresent() == true
2977         && pRaiseClaimPage.btnSubmit.isEnabled() == false) {
2978         return new RaiseClaimSuccessPage();
2979     }
2980
2981     ThreadHelper.waitForSeconds();
2982     // Mobile number
2983     pRaiseClaimPage.txtBuyerMobileEdit.click();
2984     pRaiseClaimPage.waitForElementEditable(pRaiseClaimPage.txtBuyerMobile);
2985     pRaiseClaimPage.txtBuyerMobile.clear();
2986     pRaiseClaimPage.txtBuyerMobile.sendKeys("5676757868");
2987     pRaiseClaimPage.btnSubmit.click();
2988     Assertion.assertEquals(pRaiseClaimPage.lblErrorMobileNumber.getText(),
2989         eUtil.getRED.getProperty("INVALID_MOBILE"),
2990         "Mobile number is wrong");
2991     pRaiseClaimPage.txtBuyerMobile.clear();
2992     pRaiseClaimPage.txtBuyerMobile.sendKeys("9999999999");
2993     ThreadHelper.waitForSeconds();
2994     // Landline Number
2995     pRaiseClaimPage.txtBuyerLandlineEdit.click();
2996     pRaiseClaimPage.txtBuyerLandlineEdit.click();
2997     pRaiseClaimPage.waitForElementEditable(pRaiseClaimPage.txtBuyerLandLine);
2998     pRaiseClaimPage.txtBuyerLandline.clear();
2999     pRaiseClaimPage.txtBuyerLandline.sendKeys("243561");
3000     pRaiseClaimPage.btnSubmit.click();
3001     Assertion.assertEquals(pRaiseClaimPage.lblErrLandlineNumber.getText(),
3002         eUtil.getRED.getProperty("INVALID_LANDLINE"),
3003         "Validation message for Landline number is wrong");
3004     pRaiseClaimPage.txtBuyerLandline.clear();
3005     pRaiseClaimPage.txtBuyerLandline.sendKeys("88888888");
3006
3007 ThreadHelper.waitForSeconds();
3008     if (resultMessage.equals("INR_BEFORE_ID_FAILURE")
3009         || resultMessage.equals("SNAD_NOTSHIPPED_FAILURE"))
3010

```

```

3011         return new RaiseClaimSuccessPage();
3012     } else {
3013         pRaiseClaimPage.chooseClaimType(egClaim.getClaimType());
3014         Assertion.assertEquals(
3015             pRaiseClaimPage.lblFirstClaimType.getText(),
3016             eUtil.getRED.getProperty("resultMessage"),
3017             "Error message for INR Claim Type is not matching");
3018     }
3019
3020     return new RaiseClaimSuccessPage();
3021 }
3022
3023 // SNAD_NOTSHIPPED_FAILURE
3024
3025 ThreadHelper.waitForSeconds();
3026
3027 // Invoice
3028
3029 if (pRaiseClaimPage.lblInvoice.isDisplayed() == true && !userRole.getUserRole().equalsIgnoreCase("CSR")) {
3030     pRaiseClaimPage.click();
3031     Assertion.assertEquals(pRaiseClaimPage.lblErrInvoice.getText(),
3032         eUtil.getRED.getProperty("INVALID_INVOICE"),
3033         "Invoice error message is not as expected");
3034     pRaiseClaimPage.rnYesReceivedInvoice.click();
3035 }
3036
3037 // Comments
3038 pRaiseClaimPage.txtComments.clear();
3039 pRaiseClaimPage.btnSubmit.click();
3040 Assertion.assertEquals(pRaiseClaimPage.lblErrComments.getText(),
3041         eUtil.getRED.getProperty("INVALID_COMENTS"),
3042         "Comments error message is not as expected");
3043 pRaiseClaimPage.txtComments.sendKeys(egClaim.getBuyerRemarks());
3044
3045 // Image Mandatory
3046 if (egClaim.isImageMandatory.equalsIgnoreCase("yes")) {
3047     pRaiseClaimPage.btnSubmit.click();
3048     Assertion.assertEquals(pRaiseClaimPage.lblErrUploadImage.getText(),
3049         eUtil.getRED.getProperty("INVALID_UPLOAD_IMAGE"),
3050         "Image upload error message is not as expected");
3051     if (egClaim.isUploadImage == true) {
3052         try {
3053             File file = new File("C:\\");
3054             pRaiseClaimPage.txtImageBrowse.sendKeys(file
3055                 .getCanonicalPath()
3056                 + "\\src\\main\\resources\\Image\\Hydrogeos.jpg");
3057             pRaiseClaimPage.bnUploadImage.click();
3058
3059             pRaiseClaimPage.waitForElementVisible(pRaiseClaimPage.bnImgDelete);
3060             // ThreadHelper.waitForSeconds();
3061             Assertion.assertEquals(pRaiseClaimPage.bnImgDelete
3062                 .getElements().get(0).isDisplayed(),
3063                 "Image Delete button is not displaying");
3064         } catch (Exception e) {
3065             e.printStackTrace();
3066         }
3067     }
3068 }
3069
3070

```

```

3794     } catch (Exception e) {
3795         e.printStackTrace();
3796     }
3797 }
3798 }
3799 } else if (e.getClaim().isImageMandatory().equalsIgnoreCase("no")) {
3800
3801     pRaiseClaimPage.btnSubmit.click();
3802     Assertion.assertEquals(pRaiseClaimPage.lblErrUploadImage.getText(),
3803         eUtil.eGrid.getProperty("INVALID_UPLOAD_IMAGE"),
3804         "Upload Image error message is not as expected");
3805
3806     pRaiseClaimPage.chkMandatoryImage.click();
3807 }
3808
3809 // Additional Info
3810 pRaiseClaimPage.btnAddInfo.click();
3811 Assertion.assertEquals(pRaiseClaimPage.lblErrAdditionalInfo.getText(),
3812     eUtil.eGrid.getProperty("INVALID_ADD_INFO"),
3813     "Invalid Add Info message is not as expected");
3814 pRaiseClaimPage.btnAddInfo.click();
3815 pRaiseClaimPage.btnAddInfo.click();
3816 Assertion.assertEquals(pRaiseClaimPage.lblErrOTPInfo.getText(),
3817     eUtil.eGrid.getProperty("INVALID_OTP_INFO"),
3818     "Invalid OTP Info error message is not as expected");
3819 }
3820
3821 // OTP
3822 if (UserRole.getUserRole().equalsIgnoreCase("CSR")) {
3823     pRaiseClaimPage.btnSubmit.click();
3824     Assertion.assertEquals(pRaiseClaimPage.lblErrOTPInfo.getText(),
3825         eUtil.eGrid.getProperty("INVALID_OTP_INFO"),
3826         "Invalid OTP Info error message is not as expected");
3827 }
3828
3829 /*
3830 * boolean isVerified = false; if (e.getClaim().isAlreadyOTPVerified() ==
3831 * true) {
3832 *     if (RaiseClaimOTPServices.isVerifiedOTP(userID, user.getBuyPhone1()) ==
3833 *         -- false) { String token = RaiseClaimOTPServices.sendOTP(userID,
3834 *         user.getBuyPhone1());
3835 *
3836 *         isVerified = RaiseClaimOTPServices.validateOTP(userID, token,
3837 *         EGridUtil.getOTP(userID), user.getBuyPhone1());
3838 *     }
3839 * }
3840 */
3841 if (UserRole.getUserRole().equalsIgnoreCase("CSR")) {
3842     pRaiseClaimPage.btnSubmit.click();
3843 }
3844 if (!UserRole.getUserRole().equalsIgnoreCase("CSR")) {
3845     pRaiseClaimPage.btnSubmit.click();
3846     .waitForElementToBeVisible(pRaiseClaimPage.lblOTPPopup);
3847
3848     pRaiseClaimPage.txtOTPNumber.sendKeys(EGridUtil.getOTP(userID));
3849     // pRaiseClaimPage.txtOTPNumber.sendKeys(EGridUtil.getOTP("1343500456"));
3850     ThreadHelper.waitForSeconds(5);
3851     pRaiseClaimPage.btnOTPConfirm.click();
3852 }
3853
3854 return new RaiseClaimSuccessPage();
3855
3856 }
3857 }
3858 }
3859

```

Service Util:

```

1 import com.ebay.maul.component.ebayIn.PSServices.util;
2
3 import java.io.PrintStream;
4 import java.util.ArrayList;
5 import java.util.Date;
6 import java.util.Iterator;
7 import java.util.List;
8 import java.util.Properties;
9 import java.util.Random;
10
11 import javax.xml.bind.JAXBContext;
12 import javax.xml.bind.JAXBException;
13 import javax.xml.bind.Marshaller;
14 import javax.xml.bind.annotation.XmlRootElement;
15
16 import org.openqa.selenium.By;
17
18 import com.ebay.maul.component.ebayIn.PSServicesBookShipmentStub;
19 import com.ebay.maul.component.ebayIn.PSServicesBookShipmentStub.BookShipmentRequest;
20 import com.ebay.maul.component.ebayIn.PSServicesBookShipmentStub.BookShipmentResponse;
21 import com.ebay.maul.component.ebayIn.PSServicesCheckServiceCibilityStub;
22 import com.ebay.maul.component.ebayIn.PSServicesCheckServiceCibilityStub.CheckServiceCibilityRequest;
23 import com.ebay.maul.component.ebayIn.PSServicesCommon.ConsumerDetails;
24 import com.ebay.maul.component.ebayIn.PSServicesCommon.ConsignmentDetails;
25 import com.ebay.maul.component.ebayIn.PSServicesCommon.LogisticPartner;
26 import com.ebay.maul.component.ebayIn.PSServicesCommon.PackageDetails;
27 import com.ebay.maul.component.ebayIn.PSServicesCommon.ShipperDetails;
28 import com.ebay.maul.component.ebayIn.PSServicesGetABNNumberStub;
29 import com.ebay.maul.component.ebayIn.PSServicesGetABNNumberStub.Shipment;
30 import com.ebay.maul.component.ebayIn.PSServicesGetDetailsStub;
31 import com.ebay.maul.component.ebayIn.PSServicesGetDetailsStub.AddressResponse;
32 import com.ebay.maul.component.ebayIn.PSServicesGetDetailsStub.BossDkResponse;
33 import com.ebay.maul.component.ebayIn.PSServicesGetDetailsStub.CancellationResponse;
34 import com.ebay.maul.component.ebayIn.PSServicesGetDetailsStub.PackageDetailResponse;
35 import com.ebay.maul.component.ebayIn.PSServicesGetDetailsStub.PackageDetailsResponse;
36 import com.ebay.maul.component.ebayIn.PSServicesGetDetailsStub.ReadyToShipResponse;
37 import com.ebay.maul.component.ebayIn.PSServicesGetDetailsStub.ShippingDetailResponse;
38 import com.ebay.maul.component.ebayIn.PSServicesGetDetailsStub.TrackingResponse;
39 import com.ebay.maul.component.ebayIn.common.EbayTransaction;
40 import com.ebay.maul.component.ebayIn.PSServicesGetDetailsStub.UserDetailsResponse;
41 import com.ebay.maul.component.ebayIn.common.EbayTransaction;
42 import com.ebay.maul.component.ebayIn.PSServicesScheduleShipmentsStub;
43 import com.ebay.maul.component.ebayIn.PSServicesScheduleShipmentsStub.SchedulePickupRequest;
44 import com.ebay.maul.component.ebayIn.PSServicesScheduleShipmentsStub.SchedulePickupResponse;
45 import com.ebay.maul.component.ebayIn.PSServicesScheduleShipmentsStub.ShipmentDetailsVO;
46 import com.ebay.maul.component.ebayIn.PSServicesTrackShipmentsStub;
47 import com.ebay.maul.component.ebayIn.PSServicesTrackShipmentsStub.WaybillStatusRequest;
48 import com.ebay.maul.component.ebayIn.common.EbayTransaction;
49 import com.ebay.maul.component.ebayIn.common.EbayTransaction;
50 import com.ebay.maul.component.ebayIn.common.EbayTransaction;
51 import com.ebay.maul.component.ebayIn.common.IUserRole;
52 import com.ebay.maul.component.ebayIn.eGPages.IncreaseOrderPulse;
53 import com.ebay.maul.component.ebayIn.eGPages.RaiseClaimMyCrRequest;
54 import com.ebay.maul.component.ebayIn.eGPages.SubmitClaimRequest;
55 import com.ebay.maul.component.ebayIn.eUtil.eUtil;
56 import com.ebay.maul.component.ebayIn.ps.PowerShipFms.EbayScanCode;
57 import com.ebay.maul.component.ebayIn.ps.PowerShipFms.ShippingService;
58 import com.ebay.maul.component.ebayIn.ps.LightIntegration.PSScanDetails;
59 import com.ebay.maul.component.ebayIn.ps.LightIntegration.PSTracking;
60 import com.ebay.maul.component.ebayIn.util.EbayIndiaDBUtil;
61 import com.ebay.maul.component.ebayIn.util.EbayIndiaDBUtil;
62 import com.ebay.maul.component.ebayIn.util.EbayIndiaDBUtil;
63 import com.ebay.maul.component.ebayIn.util.PowerShipUtil;
64 import com.ebay.maul.component.motors.api.APITemplate;
65 import com.ebay.maul.component.page.cal5.SellerTransactionDetailsPage;
66 import com.ebay.maul.component.page.signed.SignInPage;
67 import com.ebay.maul.component.page.xorder.XOrderConfirmationPage;

```

```
68 import com.ebay.maul.controller.Assert;
69 import com.ebay.maul.controller.ContextManager;
70 import com.ebay.maul.controller.Logging;
71 import com.ebay.maul.driver.web.WebDriver;
72 import com.ebay.maul.exception.APIException;
73 import com.ebay.maul.exception.DataCreationException;
74 import com.ebay.maul.exception.XMLException;
75 import com.ebay.maul.util.internole.entity.User;
76 import com.ebay.maul.util.sdk.eBayAccount;
77 import com.ebay.soon.eBLBaseComponents.AddItemRequestType;
78 import com.ebay.soon.eBLBaseComponents.DetailLevelType;
79 import com.ebay.soon.eBLBaseComponents.GetItemTransactionsRequestType;
80 import com.ebay.soon.eBLBaseComponents.ItemType;
81 import com.ebay.soon.eBLBaseComponents.ListingDetailsType;
82 import com.ebay.soon.eBLBaseComponents.TransactionArrayType;
83 import com.google.gson.Gson;
84 import com.google.gson.JsonBuilder;
85 import com.json.JsonSimpleDateFormat;
86 import com.sun.jersey.api.client.Client;
87 import edu.mayor.mothcs.backport.joda.util.Collections;
88
89 public class eGServiceUtil {
90     public static Properties PSServiceDetails = new Properties();
91
92     static {
93         try {
94             Class<?> c1z = Class
95                 .forName("com.ebay.maul.component.ebayIn.PSServices.util.PSServiceUtil");
96             PSServiceDetails.load(c1z
97                 .getResourcesAsStream("PSServiceDetails.properties"));
98         } catch (Exception e) {
99             e.printStackTrace();
100        }
101    }
102
103    public static void RaiseClaimRequest(EboyTransaction es, RaiseClaimSvCRequest raiseClaimRequest) throws Exception {
104
105        raiseClaimRequest.setEscrow(es.getEscrowList().get(0).getEscrowID());
106        raiseClaimRequest.setSellerContact(es.getEscrowList().get(0).getTransactionID());
107        raiseClaimRequest.setItemID(es.getEscrowList().get(0).getItemID());
108        raiseClaimRequest.setMobileNo(es.getGUll().eGcred.getProperty("BUYER_MOBILE"));
109        raiseClaimRequest.setLandlineNo(es.getGUll().eGcred.getProperty("BUYER_L1"));
110        raiseClaimRequest.setClaimType(es.getEscrowList().get(0).getEgClaim().getClaimType());
111
112        if(es.getEscrowList().get(0).getEgClaim().getPreferredResolution() != null)
113            raiseClaimRequest.setUserPref(es.getEscrowList().get(0).getEgClaim().getPreferredResolution().getResolutionID());
114
115        if(es.getEscrowList().get(0).getEgClaim().getRefundPreference() != null)
116            raiseClaimRequest.setRefundPreference(es.getEscrowList().get(0).getEgClaim().getRefundPreference().getRefundID());
117
118        raiseClaimRequest.setRemarks(es.getEscrowList().get(0).getEgClaim().getBuyerRemarks());
119        raiseClaimRequest.setSellerContactOption(es.getEscrowList().get(0).getEgClaim().getSellerContactStatus().getId());
120
121        //Added by Guanqiu
122        raiseClaimRequest.setSubIssueId(es.getEscrowList().get(0).getEgClaim().getSubIssue());
123
124        String isInvoiceReceived=es.getEscrowList().get(0).getEgClaim().isInvoiceReceived();
125        if(isInvoiceReceived.equals("YES")){
126            raiseClaimRequest.setBuyerInvoiceReceived();
127        }
128        else if(isInvoiceReceived.equals("NO")){
129            raiseClaimRequest.setBuyerInvoiceReceived();
130        }
131        else if(isInvoiceReceived.equals("N/A")){
132            raiseClaimRequest.setBuyerInvoiceReceived();
133        }
134    }
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201 }
```

```
135     raiseClaimRequest.setBuyerInvoiceReceived();
136
137     else if(isInvoiceReceived.equals("N/A")){
138         raiseClaimRequest.setBuyerInvoiceReceived();
139     }
140
141     String buyerInputImageNotRequired=es.getEscrowList().get(0).getEgClaim().isImageMandatory();
142
143     if(buyerInputImageNotRequired.equals("YES")){
144         raiseClaimRequest.setBuyerInputImageNotRequired();
145     }
146     else if(buyerInputImageNotRequired.equals("NO") || buyerInputImageNotRequired.equals("N/A")){
147         raiseClaimRequest.setBuyerInputImageNotRequired();
148     }
149
150     raiseClaimRequest.setBuyerMobileVerified(es.getEscrowList().get(0).getEgClaim().getBuyerMobileVerified());
151
152
153
154     public static void RaiseClaimResponseValidation(EboyTransaction es) throws Exception {
155         for (EboyEscrow escrow : es.getEscrowList()) {
156             EgClaim eclaim = escrow.getEgClaim();
157
158             //EboyFindJobModule.updateTransactionDate(escrow.getItem().getitemid());
159             EGClaim eGClaim = new EGClaim(escrow.getItem().getitemid());
160
161             Assertion.assertEquals(eclaim.getClaimStatus(), eGClaimDB.getClaimStatus(), "Claim status does not match");
162             Assertion.assertEquals(eclaim.getClaimType(), eGClaimDB.getClaimType(), "Claim type does not match");
163             Assertion.assertEquals(eclaim.getBuyerContactStatus(), eGClaimDB.getBuyerContactStatus(), "Contact status does not match");
164             Assertion.assertEquals(eclaim.getPreferredResolution(), eGClaimDB.getPreferredResolution(), "Preferred resolution does not match");
165             if(eclaim.getRefundPreference() != null){
166                 Assertion.assertEquals(eclaim.getRefundPreference().getRefundID(), eGClaimDB.getRefundPreference().getRefundID(), "Refund Preference not matching");
167             }
168             Assertion.assertEquals(eclaim.getClaimStage(), eGClaimDB.getClaimStage(), "Claim Stage not matching");
169             Assertion.assertEquals(eclaim.getSmirresvold(), eGClaimDB.getSmirresvold(), "SMIR not matching");
170             Assertion.assertEquals(eGClaimDB.getPriorityClaim().getid(), "Priority claim not matching");
171
172         }
173     }
174
175     public static void SubmitClaimRequest(EboyTransaction es,SubmitClaimRequest submitClaimRequest) throws Exception {
176
177         submitClaimRequest.setEscrowId(es.parseLong(es.getEscrowList().get(0).getEscrowID()));
178         submitClaimRequest.setSellerContact(es.getEscrowList().get(0).getTransactionID());
179         submitClaimRequest.setItemId(es.parseLong(es.getEscrowList().get(0).getItemID()));
180         submitClaimRequest.setClaimType(String.valueOf(es.getEscrowList().get(0).getEgClaim().getClaimType()));
181         submitClaimRequest.setRemarks("test");
182         submitClaimRequest.setBuyerRefundPreference("12408205");
183         submitClaimRequest.setActualClaimId(es.parseLong("1240867472"));
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201 }
```

```

1 package com.ebay.maul.component.ebayin.serviceutils;
2
3 import java.security.SecureRandom;
4 import javax.net.ssl.X509Certificate;
5 import java.util.List;
6 import java.util.Map;
7 import java.util.Map.Entry;
8
9 import javax.net.ssl.HostnameVerifier;
10 import javax.net.ssl.HttpsURLConnection;
11 import javax.net.ssl.SSLContext;
12 import javax.net.ssl.TrustManager;
13 import javax.net.ssl.TrustManagerFactory;
14 import javax.net.ssl.TrustManagerFactorySpi;
15 import javax.net.ssl.X509TrustManager;
16 import javax.ws.rs.core.MediaType;
17 import javax.ws.rs.core.UriBuilder;
18
19 import com.ebay.maul.controller.Logging;
20 import com.ebay.maul.util.PromisedMessageBodyWriter;
21 import com.ebay.pps.common.X509AdapterRegistry;
22 import com.ebay.pps.domain.helpers.BulletLogAdapter;
23 import com.ebay.ssm.runtime.common.config.NameValue;
24 import com.google.gson.GsonBuilder;
25
26 import com.sun.jersey.api.client.Client;
27 import com.sun.jersey.api.client.ClientResponse;
28 import com.sun.jersey.api.client.WebResource;
29 import com.sun.jersey.api.client.WebResource.Builder;
30 import com.sun.jersey.api.client.config.ClientConfig;
31 import com.sun.jersey.api.client.config.DefaultClientConfig;
32 import com.sun.jersey.json.JsonConfiguration;
33 import com.sun.jersey.client.urlconnection.HTTPSProperties;
34
35 /**
36 * @author anubhavamani
37 */
38 */
39 public class BaseServiceConsumer {
40
41     private MediaType responseMediaType = null;
42     public String baseServiceEndPoint = null;
43     public WebResource res = null;
44     private String baseServiceEndPoint = null;
45     private MediaType requestMediaType = null;
46
47
48     public static void configHttpsClient(ClientConfig config) {
49
50
51         SSLContext sc = null;
52         TrustManager[] trustAlts = new TrustManager[]{new X509TrustManager() {
53             @Override
54             public X509Certificate[] getAcceptedIssuers() { return null; }
55             @Override
56             public void checkClientTrusted(X509Certificate[] certs, String authType) {}
57         }};
58
59         try {
60             sc = SSLContext.getInstance("TLS");
61             sc.init(null, trustAlts, new SecureRandom());
62             HttpsURLConnection.setDefaultSSLSocketFactory(sc.getSocketFactory());
63         } catch (Exception e) {
64             e.printStackTrace();
65         }
66     }
67 }

```

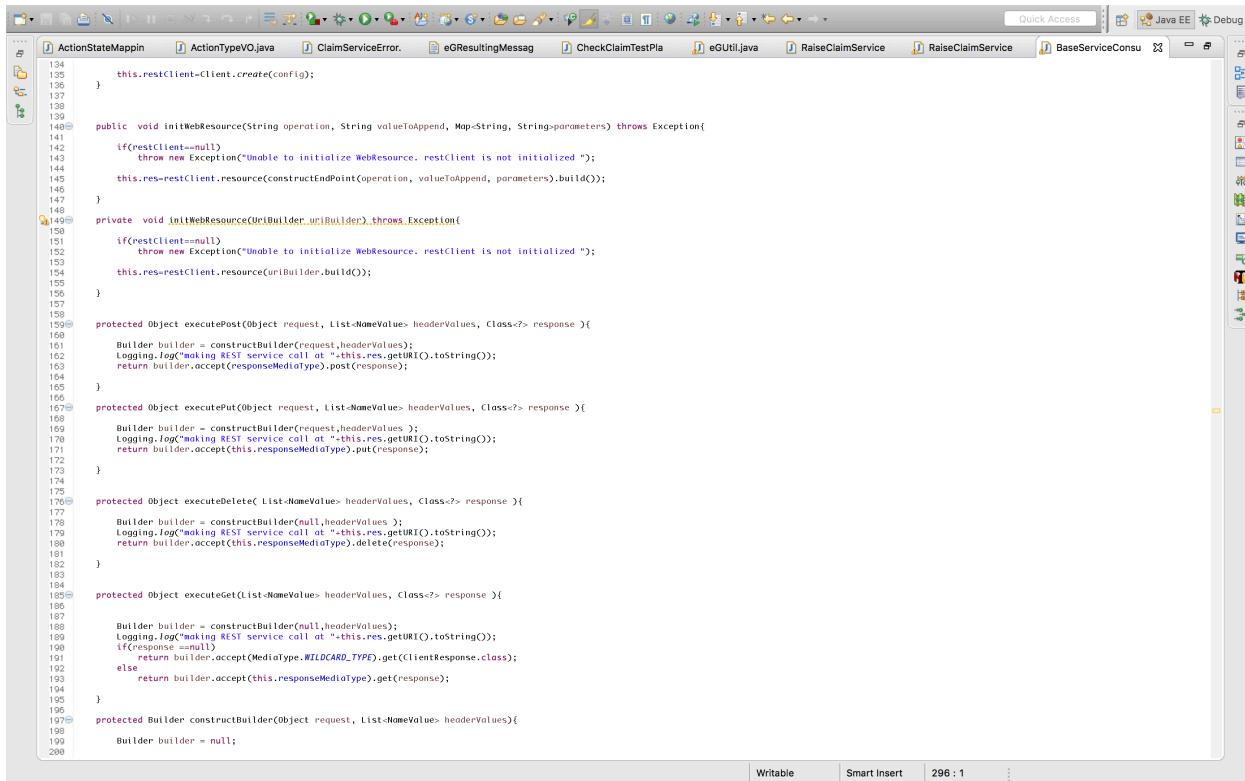
The code snippet shows the configuration of an HTTPS client. It imports various Java security and Jersey API classes. The `configHttpsClient` method initializes an `SSLContext` with a self-signed certificate trust manager. It then sets the default SSLSocketFactory to use this context.

```

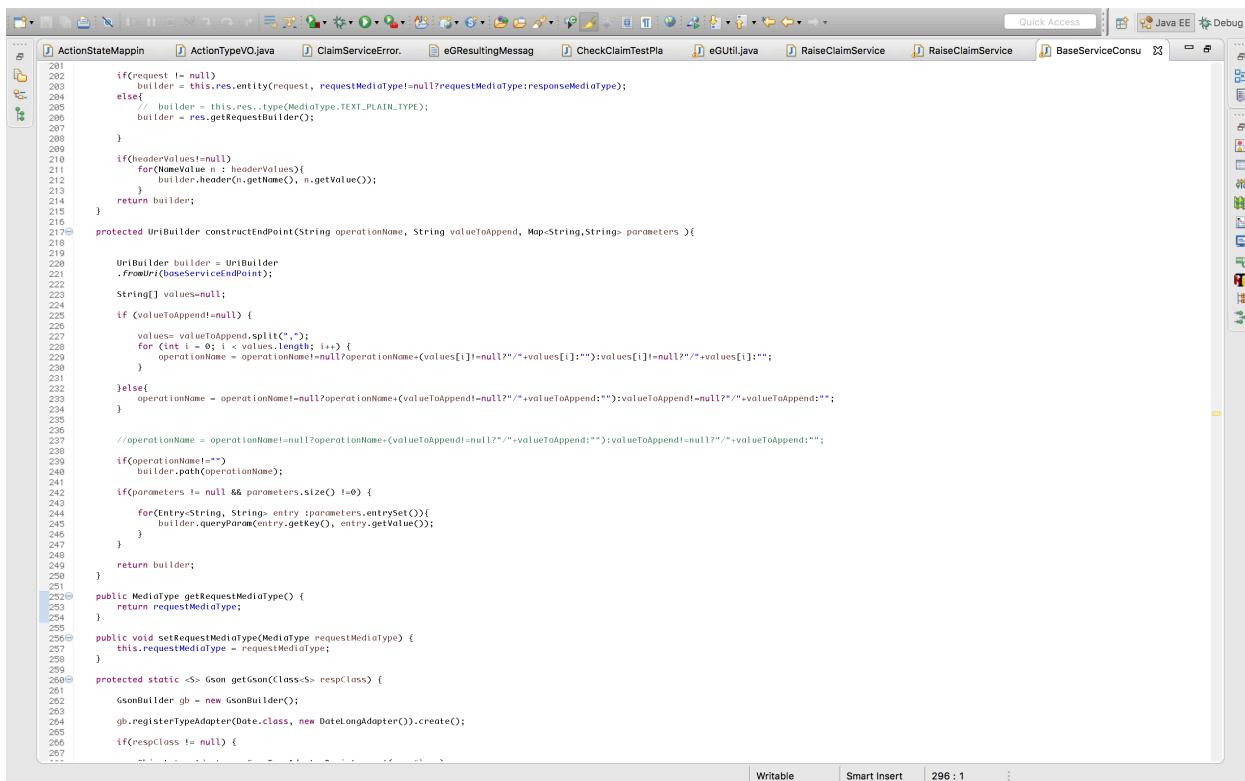
68     config.getProperties().put(HTTPSProperties.PROPERTY_HTTPS_PROPERTIES, new HTTPSProperties(new HostnameVerifier() {
69
70         //Override
71         public boolean verify(String hostname, SSLSocket session) {
72             return true;
73         }
74     }, <>));
75 }
76
77 public BaseServiceConsumer(MediaType mediatype, String baseServiceEndPoint) {
78
79     this.responseMediaType = mediatype;
80     this.baseServiceEndPoint = baseServiceEndPoint;
81     ClientConfig config = new DefaultClientConfig();
82     if(this.responseMediaType.equals(MediaType.APPLICATION_JSON_TYPE)){
83         config.getFeatures().put(JSONConfiguration.FEATURE_POJO_MAPPING, true);
84     }
85     else if(responseMediaType.getSubtype().equals("x-protobuf")){
86         config.getClasses().add(ProtobufMessageBodyReaderWriter.class);
87     }
88     this.restClient=Client.create(config);
89 }
90
91 /**
92 * @param mediatype
93 * @param baseServiceEndPoint
94 * @param requestMediaType
95 */
96 public BaseServiceConsumer(MediaType mediatype, String baseServiceEndPoint, MediaType requestMediaType ) {
97
98     this.responseMediaType = mediatype;
99     this.baseServiceEndPoint = baseServiceEndPoint;
100    this.requestMediaType = requestMediaType;
101
102    ClientConfig config = new DefaultClientConfig();
103
104    if(this.responseMediaType.equals(MediaType.APPLICATION_JSON_TYPE)){
105        config.getFeatures().put(JSONConfiguration.FEATURE_POJO_MAPPING, true);
106    }
107    else if(responseMediaType.getSubtype().equals("x-protobuf")){
108        config.getClasses().add(ProtobufMessageBodyReaderWriter.class);
109    }
110
111    this.restClient=Client.create(config);
112 }
113
114
115 public BaseServiceConsumer(MediaType responseMediaType, String baseServiceEndPoint, MediaType requestMediaType, boolean isSecuredPool ){
116
117     this.responseMediaType = responseMediaType;
118     this.baseServiceEndPoint = baseServiceEndPoint;
119     this.requestMediaType = requestMediaType;
120     ClientConfig config = new DefaultClientConfig();
121
122     if(this.responseMediaType.equals(MediaType.APPLICATION_JSON_TYPE)){
123         config.getFeatures().put(JSONConfiguration.FEATURE_POJO_MAPPING, true);
124     }
125
126     if(this.responseMediaType.getSubtype().equals("x-protobuf")){
127         config = new DefaultClientConfig();
128         config.getClasses().add(ProtobufMessageBodyReaderWriter.class);
129     }
130
131     if(isSecuredPool){
132         configHttpsClient(config);
133     }
134 }

```

The code snippet implements a `BaseServiceConsumer` class. It provides two constructors: one for standard operations and one that includes a flag for a secured pool. The secured pool constructor calls the `configHttpsClient` method to set up the HTTPS client configuration.



```
134     this.restClient=Client.create(config);
135 }
136 
137 
138 
139     public void initWebResource(String operation, String valueToAppend, Map<String, String>parameters) throws Exception{
140         if(restClient==null)
141             throw new Exception("Unable to initialize WebResource. restClient is not initialized ");
142         this.res=restClient.resource(constructEndPoint(operation, valueToAppend, parameters).build());
143     }
144 
145 
146     private void initWebResource(UriBuilder.UriBuilder) throws Exception{
147         if(restClient==null)
148             throw new Exception("Unable to initialize WebResource. restClient is not initialized ");
149         this.res=restClient.resource(uriBuilder.build());
150     }
151 
152 
153     protected Object executePost(Object request, List<NameValuePair> headerValues, Class<?> response ){
154         Builder builder = constructBuilder(request,headerValues);
155         Logging.Log("making REST service call at "+this.res.getURI().toString());
156         return builder.accept(responseMediaType).post(response);
157     }
158 
159 
160     protected Object executePut(Object request, List<NameValuePair> headerValues, Class<?> response ){
161         Builder builder = constructBuilder(request,headerValues );
162         Logging.Log("making REST service call at "+this.res.getURI().toString());
163         return builder.accept(this.responseMediaType).put(response);
164     }
165 
166 
167     protected Object executeDelete( List<NameValuePair> headerValues, Class<?> response ){
168         Builder builder = constructBuilder(null,headerValues );
169         Logging.Log("making REST service call at "+this.res.getURI().toString());
170         return builder.accept(this.responseMediaType).delete(response);
171     }
172 
173 
174     protected Object executeGet(List<NameValuePair> headerValues, Class<?> response ){
175         Builder builder = constructBuilder(null,headerValues);
176         Logging.Log("making REST service call at "+this.res.getURI().toString());
177         if(response ==null)
178             return builder.accept(MediaType.WILDCARD_TYPE).get(ClientResponse.class);
179         else
180             return builder.accept(this.responseMediaType).get(response);
181     }
182 
183 
184     protected Builder constructBuilder(Object request, List<NameValuePair> headerValues){
185         Builder builder = null;
186     }
187 
```



```
201         if(request != null)
202             builder = this.res.entityRequest( requestMediaType!=null?requestMediaType:responseMediaType );
203         else
204             builder = this.res.type(MediaType.TEXT_PLAIN_TYPE);
205         builder = res.getRequestBuilder();
206     }
207 
208 
209     if(headerValues!=null)
210         for(NameValuePair n : headerValues){
211             builder.header(n.getName(), n.getValue());
212         }
213     return builder;
214 }
215 
216 
217     protected UriBuilder constructEndPoint(String operationName, String valueToAppend, Map<String, String> parameters ){
218 
219         UriBuilder builder = UriBuilder
220             .fromUri(baseServiceEndPoint);
221 
222         String[] values=null;
223 
224         if (valueToAppend!=null) {
225 
226             values= valueToAppend.split(",");
227             for (int i = 0; i < values.length; i++) {
228                 operationName = operationName+ (values[i]==null?"":values[i]);values[i]==null?"":values[i];
229             }
230         }
231 
232         if(values!=null) {
233             for (int i = 0; i < values.length; i++) {
234                 operationName = operationName+ (values[i]==null?"":values[i]);values[i]==null?"":values[i];
235             }
236         }
237 
238         //operationName = operationName+ (valueToAppend==null?"":valueToAppend);valueToAppend==null?"":valueToAppend;
239 
240         if(operationName=="")
241             builder.path(operationName);
242 
243         if(parameters != null && parameters.entrySet().size() !=0) {
244 
245             for(Entry<String, String> entry:parameters.entrySet()){
246                 builder.queryParam(entry.getKey(), entry.getValue());
247             }
248         }
249 
250         return builder;
251     }
252 
253     public MediaType getRequestMediaType() {
254         return requestMediaType;
255     }
256 
257     public void setRequestMediaType(MediaType requestMediaType) {
258         this.requestMediaType = requestMediaType;
259     }
260 
261     protected static <S> Gson getGson(Class<S> respClass) {
262 
263         GsonBuilder gb = new GsonBuilder();
264 
265         gb.registerTypeAdapter(Date.class, new DateLongAdapter()).create();
266 
267         if(respClass != null) {
268             ...
269         }
270     }
271 
```

The screenshot shows a Java code editor in an IDE. The code is part of the `ActionStateMapper` class. It includes logic for building query parameters based on operation name and value-to-append pairs, handling null values, and setting request media types. The code uses annotations like `@Path`, `@QueryParam`, and `@RequestHeader`. It also includes a static method for creating a Gson builder and a main method for testing the consumer.

```
230     }
231     else{
232         operationName = operationName!=null?operationName+(valueToAppend!=null?"/"+valueToAppend:""):(valueToAppend!=null?"/"+valueToAppend:"");
233     }
234     }
235     //operationName = operationName!=null?operationName+(valueToAppend!=null?"/"+valueToAppend:""):(valueToAppend!=null?"/"+valueToAppend:"");
236     if(operationName!="")
237         builder.path(operationName);
238     }
239     if(parameters != null && parameters.size() !=0) {
240         for(Entry<String, String> entry :parameters.entrySet()){
241             builder.queryParam(entry.getKey(), entry.getValue());
242         }
243     }
244     return builder;
245 }
246 public MediaType getRequestMediaType() {
247     return requestMediaType;
248 }
249 public void setRequestMediaType(MediaType requestMediaType) {
250     this.requestMediaType = requestMediaType;
251 }
252 protected static < T> Gson getGson(Class< T> respClass) {
253     GsonBuilder gb = new GsonBuilder();
254     gb.registerTypeAdapter(Date.class, new DateLongAdapter()).create();
255     if(respClass != null) {
256         Object typeAdapter = GsonTypeAdapterRegistry.get(respClass);
257         if(typeAdapter != null) {
258             gb.registerTypeAdapter(respClass, typeAdapter);
259         }
260     }
261     return gb.create();
262 }
263 */
264 public static void main(String [] args) throws Exception{
265     Map<String, String> t = new HashMap<String, String>();
266     t.put("1", "10");
267     t.put("2", "1");
268     BaseServiceConsumer consumer = new BaseServiceConsumer();
269     consumer.baseServiceEndPoint = "www.gp.ebay.com";
270     Uri build = consumer.constructEndPoint("service","test",t).build();
271     System.out.println(build.toString());
272 }
273 */
274 }
```

The screenshot shows a Java code editor in an IDE. The code is part of the `RaiseClaimServiceConsumer` class. It defines a base service endpoint and implements methods for raising a claim and claiming a raise. The implementation uses `Gson` for JSON parsing and `PowerShipUtil` for configuration. It handles `MediaType.APPLICATION_JSON_TYPE` and `MediaType.APPLICATION_XML_TYPE`.

```
1 package com.ebay.moui.component.ebayIn.et.servicePages;
2
3 import javax.ws.rs.core.MediaType;
4
5 import com.ebay.moui.component.ebayIn.et.servicePages.ActiveValue;
6 import com.ebay.moui.component.ebayIn.et.servicePages.ActiveClaimSvcRequest;
7 import com.ebay.moui.component.ebayIn.et.servicePages.RaiseClaimSvcResponse;
8 import com.ebay.moui.component.ebayInUtils.PowerShipUtil;
9 import com.google.gson.Gson;
10
11 public class RaiseClaimServiceConsumer extends BaseServiceConsumer{
12     public static final String BASE_SERVICE_END_POINT = PowerShipUtil.adminCred.getProperty("eGClaimService");
13
14     final static String RAISE_CLAIM = "raiseClaim";
15
16     public RaiseClaimServiceConsumer(){
17         super(MediaType.APPLICATION_JSON_TYPE, BASE_SERVICE_END_POINT,MediaType.APPLICATION_XML_TYPE );
18     }
19
20     public RaiseClaimServiceConsumer(){
21         super(MediaType.APPLICATION_JSON_TYPE, BASE_SERVICE_END_POINT,MediaType.APPLICATION_XML_TYPE );
22     }
23
24     public void raiseClaim(RaiseClaimSvcRequest request ) throws Exception{
25         initWebResource(RAISE_CLAIM, null, null);
26         ClientResponse response = client().post(Entity.entity(request, null, String.class));
27         Gson gson = getGson(RaiseClaimSvcResponse.class);
28         RaiseClaimSvcResponse fromJson = gson.fromJson(response, RaiseClaimSvcResponse.class);
29         System.out.println(fromJson.getClaimReqId());
30         System.out.println(fromJson.getErrors().get(0).getErrorCode());
31         System.out.println(fromJson.getErrors().get(0).getErrorMessage());
32     }
33
34     public String claimRaise(RaiseClaimSvcRequest request ) throws Exception{
35         initWebResource(RAISE_CLAIM, null, null);
36         String response = (String)this.executePost(request, null, String.class);
37         GsonBuilder gb = new GsonBuilder();
38         gb.registerTypeAdapter(RaiseClaimSvcResponse.class, new RaiseClaimSvcResponse.GsonTypeAdapter());
39         RaiseClaimSvcResponse fromJson = gson.fromJson(response, RaiseClaimSvcResponse.class);
40         if(fromJson.getAck().name().equals("SUCCESS")){
41             return String.valueOf(fromJson.getClaimReqId());
42         }else{
43             return null;
44         }
45     }
46 }
```