

## Controller:



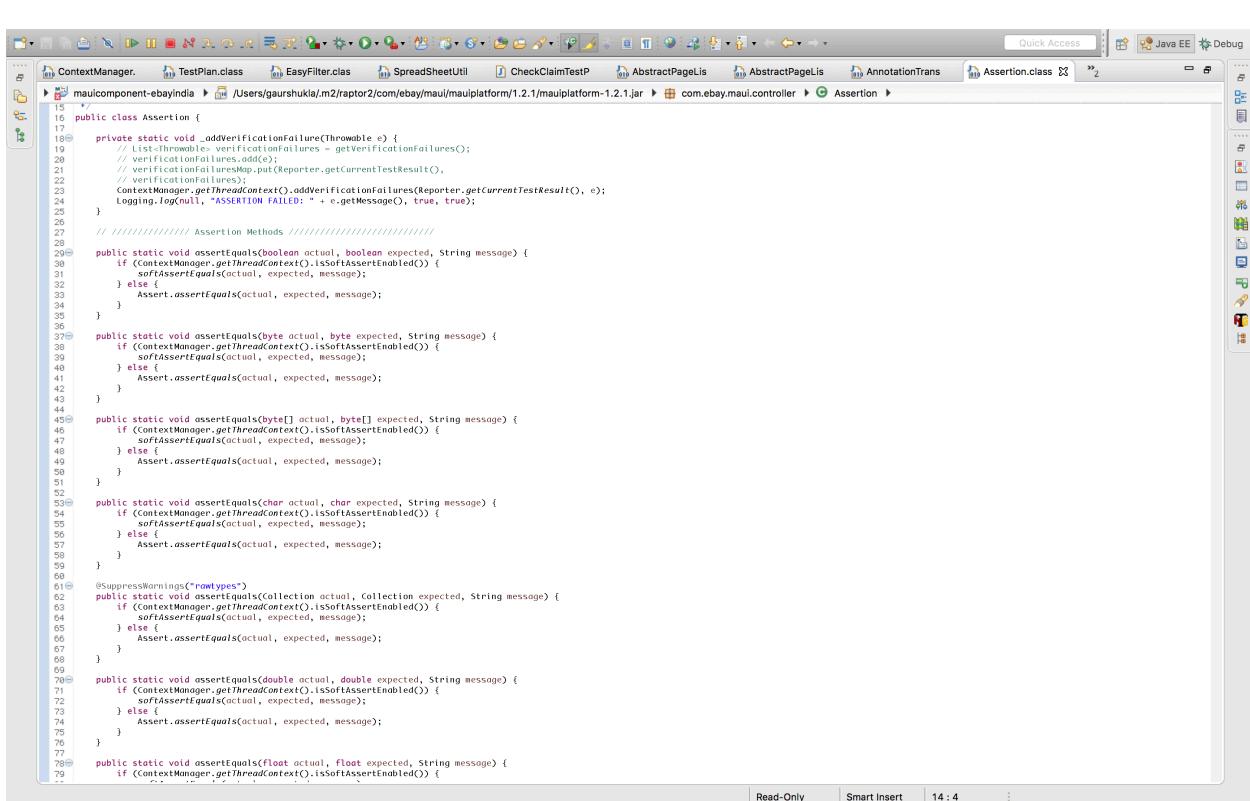
The screenshot shows the Eclipse IDE interface with the Java EE perspective selected. The top bar includes the 'Quick Access' button and the 'Java EE' icon. The left sidebar displays the package structure under 'com.ebay.maui.controller'. The main editor area contains the 'AbstractPageListener.java' code, which is annotated with numerous Javadoc comments. The code defines an abstract class with various methods for handling page load and unload events, as well as static utility methods for opening URLs and setting test results.

```
1 public abstract class AbstractPageListener {
2     protected static final Logger logger = Logging.getLogger(AbstractPageListener.class);
3
4     /**
5      * Notifies all the page listeners on page Load
6      *
7      * @param page
8      */
9     public static void notifyPageLoad(IPage page) {
10         PluginUtil.getInstance().invokePageListeners(ContextHolder.getThreadContext(), getTestMethodSignature(),
11             page, true);
12     }
13
14     /**
15      * Notifies all the page listeners on page Unload
16      *
17      * @param page
18      */
19     public static void notifyPageUnload(IPage page) {
20         PluginUtil.getInstance().invokePageListeners(ContextHolder.getThreadContext(), getTestMethodSignature(),
21             page, false);
22     }
23
24     private String title;
25
26     private boolean testResultEffectected;
27
28     public AbstractPageListener(String title, boolean testResultEffectected) {
29         this.title = title;
30         this.testResultEffectected = testResultEffectected;
31     }
32
33     public AbstractPageListener() {
34     }
35
36     public String getTitle() {
37         return title;
38     }
39
40     public boolean isTestResultEffectected() {
41         return testResultEffectected;
42     }
43
44     public abstract void onPageLoad(IPage page);
45
46     public abstract void onPageUnload(IPage page);
47
48     //////////////////// Static Helpers /////////////////////
49
50     protected String openURL(String url) throws Exception {
51         return URLHelper.openUrl(url);
52     }
53
54     public void setTestResultEffectected(boolean testResultEffectected) {
55         this.testResultEffectected = testResultEffectected;
56     }
57
58     public void setTitle(String title) {
59         this.title = title;
60     }
61 }
```

```

35  public String getInvocationTimeout() {
36      return invocationTimeout;
37  }
38
39  public void setInvocationTimeout(String invocationTimeout) {
40      this.invocationTimeout = invocationTimeout;
41  }
42
43  public String getThreadPoolSize() {
44      return threadPoolSize;
45  }
46
47  public void setThreadPoolSize(String threadPoolSize) {
48      this.threadPoolSize = threadPoolSize;
49  }
50
51  public String getTestMethodName() {
52      return testMethodName;
53  }
54
55  public void setTestMethodName(String testMethodName) {
56      this.testMethodName = testMethodName;
57  }
58
59  public AnnotationTransformer() {
60      invocationCount = System.getProperty(INVOCATION_COUNT);
61      invocationTimeout = System.getProperty(INVOCATION_TIME_OUT);
62      threadPoolSize = System.getProperty(THREAD_POOL_SIZE);
63      testMethodName = System.getProperty(TEST_METHOD_NAME);
64  }
65
66  @SuppressWarnings("rawtypes")
67  public void transformObjectAnnotation(annotation, Class testClass, Constructor testConstructor, Method testMethod) {
68      if (testMethodName != null) {
69          if (testMethod != null && testMethod.getName().contains(testMethodName)) {
70              if (invocationCount != null) {
71                  annotation.setInvocationCount(Integer.parseInt(invocationCount));
72              }
73
74              if (invocationTimeout != null) {
75                  annotation.setInvocationTimeout(Long.parseLong(invocationTimeout));
76              }
77
78              if (threadPoolSize != null) {
79                  annotation.setThreadPoolSize(Integer.parseInt(threadPoolSize));
80              }
81          } else {
82              if (invocationCount != null) {
83                  annotation.setInvocationCount(Integer.parseInt(invocationCount));
84              }
85
86              if (invocationTimeout != null) {
87                  annotation.setInvocationTimeout(Long.parseLong(invocationTimeout));
88              }
89
90              if (threadPoolSize != null) {
91                  annotation.setThreadPoolSize(Integer.parseInt(threadPoolSize));
92              }
93          }
94      }
95  }
96
97 }
98

```



The screenshot shows an IDE interface with the Assertion class open. The code is identical to the one shown in the previous screenshot, but the IDE provides syntax highlighting and navigation tools. The top bar includes tabs for Java EE and Debug, and the bottom status bar shows 'Read-Only' and 'Smart Insert'.

```

15  public class Assertion {
16
17      private static void addVerificationFailure(Throwable e) {
18          // List<Throwable> verificationFailures = getVerificationFailures();
19          // verificationFailures.add(e);
20          // verificationFailuresMap.put(Reporter.getCurrentTestResult(),
21          // verificationFailures);
22          ContextManager.getThreadContext().addVerificationFailures(Reporter.getCurrentTestResult(), e);
23          Logging.logNull("ASSERTION FAILED: " + e.getMessage(), true, true);
24      }
25
26      // //////////////////// Assertion Methods ///////////////////
27
28      public static void assertEquals(boolean actual, boolean expected, String message) {
29          if (ContextManager.getThreadContext().isSoftAssertEnabled()) {
30              softAssertEquals(actual, expected, message);
31          } else {
32              Assert.assertEquals(actual, expected, message);
33          }
34      }
35
36      public static void assertEquals(byte actual, byte expected, String message) {
37          if (ContextManager.getThreadContext().isSoftAssertEnabled()) {
38              softAssertEquals(actual, expected, message);
39          } else {
40              Assert.assertEquals(actual, expected, message);
41          }
42      }
43
44      public static void assertEquals(byte[] actual, byte[] expected, String message) {
45          if (ContextManager.getThreadContext().isSoftAssertEnabled()) {
46              softAssertEquals(actual, expected, message);
47          } else {
48              Assert.assertEquals(actual, expected, message);
49          }
50      }
51
52      public static void assertEquals(char actual, char expected, String message) {
53          if (ContextManager.getThreadContext().isSoftAssertEnabled()) {
54              softAssertEquals(actual, expected, message);
55          } else {
56              Assert.assertEquals(actual, expected, message);
57          }
58      }
59
60      @SuppressWarnings("rawtypes")
61      public static void assertEquals(Collection actual, Collection expected, String message) {
62          if (ContextManager.getThreadContext().isSoftAssertEnabled()) {
63              softAssertEquals(actual, expected, message);
64          } else {
65              Assert.assertEquals(actual, expected, message);
66          }
67      }
68
69      public static void assertEquals(double actual, double expected, String message) {
70          if (ContextManager.getThreadContext().isSoftAssertEnabled()) {
71              softAssertEquals(actual, expected, message);
72          } else {
73              Assert.assertEquals(actual, expected, message);
74          }
75      }
76
77      public static void assertEquals(float actual, float expected, String message) {
78          if (ContextManager.getThreadContext().isSoftAssertEnabled()) {
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99

```

```

 78    public static void assertEquals(float actual, float expected, String message) {
 79      if (ContextManager.getThreadContext().isSoftAssertEnabled()) {
 80        softAssertEquals(actual, expected, message);
 81      } else {
 82        Assert.assertEquals(actual, expected, message);
 83      }
 84    }
 85
 86    public static void assertEquals(int actual, int expected, String message) {
 87      if (ContextManager.getThreadContext().isSoftAssertEnabled()) {
 88        softAssertEquals(actual, expected, message);
 89      } else {
 90        Assert.assertEquals(actual, expected, message);
 91      }
 92    }
 93
 94    public static void assertEquals(long actual, long expected, String message) {
 95      if (ContextManager.getThreadContext().isSoftAssertEnabled()) {
 96        softAssertEquals(actual, expected, message);
 97      } else {
 98        Assert.assertEquals(actual, expected, message);
 99      }
 100    }
 101
 102    public static void assertEquals(Object actual, Object expected, String message) {
 103      if (ContextManager.getThreadContext().isSoftAssertEnabled()) {
 104        softAssertEquals(actual, expected, message);
 105      } else {
 106        Assert.assertEquals(actual, expected, message);
 107      }
 108    }
 109
 110    public static void assertEquals(Short actual, Short expected, String message) {
 111      if (ContextManager.getThreadContext().isSoftAssertEnabled()) {
 112        softAssertEquals(actual, expected, message);
 113      } else {
 114        Assert.assertEquals(actual, expected, message);
 115      }
 116    }
 117
 118    public static void assertEquals(String[] actual, String[] expected, String message) {
 119      if (ContextManager.getThreadContext().isSoftAssertEnabled()) {
 120        softAssertEquals(actual, expected, message);
 121      } else {
 122        Assert.assertEquals(actual, expected, message);
 123      }
 124    }
 125
 126    public static void assertEquals(String actual, String expected, String message) {
 127      if (ContextManager.getThreadContext().isSoftAssertEnabled()) {
 128        softAssertEquals(actual, expected, message);
 129      } else {
 130        Assert.assertEquals(actual, expected, message);
 131      }
 132    }
 133
 134    public static void assertFalse(boolean condition, String message) {
 135      if (ContextManager.getThreadContext().isSoftAssertEnabled()) {
 136        softAssertFalse(condition, message);
 137      } else {
 138        Assert.assertFalse(condition, message);
 139      }
 140    }
 141
 142    public static void assertNotNull(Object object, String message) {
 143
 144      if (ContextManager.getThreadContext().isSoftAssertEnabled()) {
 145        softAssertNotNull(object, message);
 146      } else {
 147        Assert.assertNotNull(object, message);
 148      }
 149    }
 150
 151    public static void assertNotSame(Object actual, Object expected, String message) {
 152      if (ContextManager.getThreadContext().isSoftAssertEnabled()) {
 153        softAssertNotSame(actual, expected, message);
 154      } else {
 155        Assert.assertNotSame(actual, expected, message);
 156      }
 157    }
 158
 159    public static void assertNull(Object object, String message) {
 160      if (ContextManager.getThreadContext().isSoftAssertEnabled()) {
 161        softAssertNull(object, message);
 162      } else {
 163        Assert.assertNull(object, message);
 164      }
 165    }
 166
 167    public static void assertSame(Object actual, Object expected, String message) {
 168      if (ContextManager.getThreadContext().isSoftAssertEnabled()) {
 169        softAssertSame(actual, expected, message);
 170      } else {
 171        Assert.assertSame(actual, expected, message);
 172      }
 173    }
 174
 175    public static void assertTrue(boolean condition, String message) {
 176      if (ContextManager.getThreadContext().isSoftAssertEnabled()) {
 177        softAssertTrue(condition, message);
 178      } else {
 179        Assert.assertTrue(condition, message);
 180      }
 181    }
 182
 183    public static void fail(String message) {
 184      Assert.fail(message);
 185    }
 186
 187    public static List<Throwable> getVerificationFailures() {
 188      // List<Throwable> verificationFailures =
 189      // verificationFailuresMap.get(Reporter.getCurrentTestResult());
 190      // return verificationFailures == null ? new ArrayList<Throwable>() :
 191      // verificationFailures;
 192      return ContextManager.getThreadContext().getVerificationFailures(Reporter.getCurrentTestResult());
 193    }
 194
 195    // ////////////////// Soft Assertion Methods ///////////////////
 196
 197    public static void softAssertEquals(boolean actual, boolean expected, String message) {
 198
 199      try {
 200        Assert.assertEquals(actual, expected, message);
 201      } catch (Throwable e) {
 202        addVerificationFailure(e);
 203      }
 204    }
 205

```

```

209
210     public static void softAssertEquals(byte actual, byte expected, String message) {
211         try {
212             Assert.assertEquals(actual, expected, message);
213         } catch (Throwable e) {
214             _addVerificationFailure(e);
215         }
216     }
217     public static void softAssertEquals(byte[] actual, byte[] expected, String message) {
218         try {
219             Assert.assertEquals(actual, expected, message);
220         } catch (Throwable e) {
221             _addVerificationFailure(e);
222         }
223     }
224     public static void softAssertEquals(char actual, char expected, String message) {
225         try {
226             Assert.assertEquals(actual, expected, message);
227         } catch (Throwable e) {
228             _addVerificationFailure(e);
229         }
230     }
231     @SuppressWarnings("rawtypes")
232     public static void softAssertEquals(Collection actual, Collection expected, String message) {
233         try {
234             Assert.assertEquals(actual, expected, message);
235         } catch (Throwable e) {
236             _addVerificationFailure(e);
237         }
238     }
239     public static void softAssertEquals(double actual, double expected, String message) {
240         try {
241             Assert.assertEquals(actual, expected, message);
242         } catch (Throwable e) {
243             _addVerificationFailure(e);
244         }
245     }
246     public static void softAssertEquals(float actual, float expected, String message) {
247         try {
248             Assert.assertEquals(actual, expected, message);
249         } catch (Throwable e) {
250             _addVerificationFailure(e);
251         }
252     }
253     public static void softAssertEquals(int actual, int expected, String message) {
254         try {
255             Assert.assertEquals(actual, expected, message);
256         } catch (Throwable e) {
257             _addVerificationFailure(e);
258         }
259     }
260     public static void softAssertEquals(long actual, long expected, String message) {
261         try {
262             Assert.assertEquals(actual, expected, message);
263         } catch (Throwable e) {
264             _addVerificationFailure(e);
265         }
266     }
267     public static void softAssertEquals(Object actual, Object expected, String message) {
268         try {
269             Assert.assertEquals(actual, expected, message);
270         } catch (Throwable e) {
271             _addVerificationFailure(e);
272         }
273     }
274     public static void softAssertEquals(Object[] actual, Object[] expected, String message) {
275         try {
276             Assert.assertEquals(actual, expected, message);
277         } catch (Throwable e) {
278             _addVerificationFailure(e);
279         }
280     }
281     public static void softAssertEquals(short actual, short expected, String message) {
282         try {
283             Assert.assertEquals(actual, expected, message);
284         } catch (Throwable e) {
285             _addVerificationFailure(e);
286         }
287     }
288     public static void softAssertEquals(String actual, String expected, String message) {
289         try {
290             Assert.assertEquals(actual, expected, message);
291         } catch (Throwable e) {
292             _addVerificationFailure(e);
293         }
294     }

```

The screenshot shows the Assertion class code in an IDE. The code is annotated with line numbers from 311 to 395. It contains several static methods for performing assertions:

```

311     }
312   }
313 
314   public static void softAssertFalse(boolean condition, String message) {
315     try {
316       Assert.assertFalse(condition, message);
317     } catch (Throwable e) {
318       _addVerificationFailure(e);
319     }
320   }
321 
322   public static void softAssertNotNull(Object object, String message) {
323     try {
324       Assert.assertNotNull(object, message);
325     } catch (Throwable e) {
326       _addVerificationFailure(e);
327     }
328   }
329 
330   public static void softAssertNotSame(Object actual, Object expected, String message) {
331     try {
332       Assert.assertNotSame(actual, expected, message);
333     } catch (Throwable e) {
334       _addVerificationFailure(e);
335     }
336   }
337 
338   public static void softAssertNull(Object object, String message) {
339     try {
340       Assert.assertNull(object, message);
341     } catch (Throwable e) {
342       _addVerificationFailure(e);
343     }
344   }
345 
346   public static void softAssertSame(Object actual, Object expected, String message) {
347     try {
348       Assert.assertSame(actual, expected, message);
349     } catch (Throwable e) {
350       _addVerificationFailure(e);
351     }
352   }
353 
354   public static void softAssertTrue(boolean condition, String message) {
355     try {
356       Assert.assertTrue(condition, message);
357     } catch (Throwable e) {
358       _addVerificationFailure(e);
359     }
360   }
361 
362   public static void softAssertTrue(boolean condition, String message) {
363     try {
364       Assert.assertTrue(condition, message);
365     } catch (Throwable e) {
366       _addVerificationFailure(e);
367     }
368   }
369 
370   public static void initGlobalContext(ITestContext testContext) {
371     try {
372       _initGlobalContext(testContext);
373     } catch (Exception e) {
374       _addVerificationFailure(e);
375     }
376   }
377 
378   public static Context getGlobalContext() {
379     return _globalContext;
380   }
381 
382   public static Context getTestLevelContext() {
383     return _testLevelContext;
384   }
385 
386   public static Context getThreadLocalContext() {
387     return _threadLocalContext;
388   }
389 
390   public static void addContextAttributeListener(IContextAttributeListener listener) {
391     _contextAttributeListenerList.add(listener);
392   }
393 
394 }
395 
```

## Context:

The screenshot shows the ContextManager class code in an IDE. The code is annotated with line numbers from 23 to 87. It defines various context levels and initializes them:

```

23 public class ContextManager {
24   // Customized Context Attribute
25   private static List<IContextAttributeListener> _contextAttributeListenerList = Collections.synchronizedList(new ArrayList<IContextAttributeListener>());
26 
27   // define the global level context
28   private static Context _globalContext;
29 
30   // define the test level context
31   private Map<String, Context> _testLevelContext = Collections.synchronizedMap(new HashMap<String, Context>());
32 
33   // define the thread level context
34   private static ThreadLocal<Context> _threadLocalContext = new ThreadLocal<Context>();
35 
36   public static void addContextAttributeListener(IContextAttributeListener listener) {
37     _contextAttributeListenerList.add(listener);
38   }
39 
40   public static Context getGlobalContext() {
41     if (_globalContext == null)
42       System.out.println("Initialize default GlobalContext");
43     _initGlobalContext(new DefaultTestNGContext());
44     return _globalContext;
45   }
46 
47   public static Context getTestLevelContext(ITestContext testContext) {
48     if (testContext != null && testContext.getCurrentXmlTest() != null) {
49       if (_testLevelContext.get(testContext.getCurrentXmlTest().getName()) == null) {
50         // sometimes getTestLevelContext is called before @BeforeTest
51         _testLevelContext.initTestLevelContext(testContext, testContext.getCurrentXmlTest());
52       }
53       return _testLevelContext.get(testContext.getCurrentXmlTest().getName());
54     } else {
55       return null;
56     }
57   }
58 
59   public static Context getTestLevelContext(String testName) {
60     return _testLevelContext.get(testName);
61   }
62 
63   public static Context getThreadContext() {
64     if (_threadLocalContext.get() == null)
65       System.out.println("Initialize default ThreadContext");
66     _initThreadContext(_globalContext.getTestNGContext(), null);
67     return _threadLocalContext.get();
68   }
69 
70   public static void initGlobalContext(ITestContext testNGCtx) {
71     testNGCtx = getTestContextFromConfigFile(testNGCtx);
72     _globalContext = new Context(testNGCtx);
73     loadCustomizedContextAttribute(testNGCtx, _globalContext);
74   }
75 
76   private static ITestContext getContextFromConfigFile(ITestContext testContext) {
77     if (testContext != null) {
78       if (testContext.getSuite() != null && testContext.getSuite().getXmlSuite() != null) {
79         File suiteFile = testContext.getSuite().getXmlSuite().getFile();
80         String configPath = suiteFile.getPath().replace(suiteFile.getName(), "") + testContext.getSuite().getParameters().get("testConfig");
81         NodeList nList = XMLHelper.getXMLNodes(configPath);
82         Map<String, String> parameters = testContext.getSuite().getXmlSuite().getParameters();
83         for (int i = 0; i < nList.getLength(); i++) {
84           Node node = nList.item(i);
85           if (node instanceof Element) {
86             Element element = (Element) node;
87             String name = element.getAttribute("name");
88             String value = element.getAttribute("value");
89             parameters.put(name, value);
90           }
91         }
92       }
93     }
94     return testContext;
95   }
96 
```

```

 78 }
 79 
 80     private static ITestContext getContextFromConfigFile(ITestContext testContext) {
 81         if (testContext != null) {
 82             if ((testContext.getSuite().getParameters().get("TEST_CONFIG") != null) &
 83                 (!testContext.getSuite().getParameters().get("TEST_CONFIG").getFileName().equals("maulplatform"))){
 84                 String configFile = XMLHelper.getNodes(configFile, "parameter");
 85                 Map<String, String> testParameters = testContext.getSuite().getParameters();
 86                 for (Node node : configFile.getLength(); i++) {
 87                     Node nNode = nList.item(i);
 88                     parameters.put(nNode.getAttributes().getNamedItem("name").getNodeValue(), nNode.getAttributes().getNamedItem("value").getNodeValue());
 89                 }
 90             }
 91             testContext.getSuite().getXmlSuite().setParameters(parameters);
 92         }
 93     }
 94     return testContext;
 95 }
 96 
 97     public static void initTestLevelContext(ITestContext testNGCtx, XmlTest xmlTest) {
 98         Context maulCtx = new Context(testNGCtx);
 99         if (ComTest != null) {
100             Map<String, String> testParameters = xmlTest.getTestParameters();
101             // parse the test level parameters
102             for (Entry<String, String> entry : testParameters.entrySet()) {
103                 maulCtx.setAttribute(entry.getKey(), entry.getValue());
104             }
105         }
106         testLevelContext.put(xmlTest.getName(), maulCtx);
107     }
108 
109     public static void initTestLevelContext(XmlTest xmlTest) {
110         initTestLevelContext(globalContext.getTestNGContext(), xmlTest);
111     }
112 
113     public static void initThreadContext() {
114         initThreadContext(globalContext.getTestNGContext(), null);
115     }
116 
117     public static void initThreadContext(ITestContext testNGCtx) {
118         initThreadContext(testNGCtx, null);
119     }
120 
121     public static void initThreadContext(ITestContext testNGCtx, XmlTest xmlTest) {
122         Context maulCtx = new Context(testNGCtx);
123         loadCustomizedContextAttribute(testNGCtx, maulCtx);
124 
125         if (ComTest != null) {
126             Map<String, String> testParameters = xmlTest.getTestParameters();
127             // parse the test level parameters
128             for (Entry<String, String> entry : testParameters.entrySet()) {
129                 if (System.getProperty(entry.getKey()) == null)
130                     maulCtx.setAttribute(entry.getKey(), entry.getValue());
131             }
132         }
133         threadLocalContext.set(maulCtx);
134     }
135 
136     public static void initThreadContext(XmlTest xmlTest) {
137 
138 }
139 
140     public static void initTestLevelContext(ITestContext testNGCtx, XmlTest xmlTest) {
141         Context maulCtx = new Context(testNGCtx);
142         if (ComTest != null) {
143             Map<String, String> testParameters = xmlTest.getTestParameters();
144             // parse the test level parameters
145             for (Entry<String, String> entry : testParameters.entrySet()) {
146                 if (System.getProperty(entry.getKey()) == null)
147                     maulCtx.setAttribute(entry.getKey(), entry.getValue());
148             }
149         }
150         testLevelContext.put(xmlTest.getName(), maulCtx);
151     }
152 
153     public static void initTestLevelContext(XmlTest xmlTest) {
154         initTestLevelContext(globalContext.getTestNGContext(), xmlTest);
155     }
156 
157     private static void loadCustomizedContextAttribute(ITestContext testNGCtx, Context maulCtx) {
158         for (int i = 0; i < contextXmlAttributeList.size(); i++) {
159             contextXmlAttributeList.get(i).load(testNGCtx, maulCtx);
160         }
161     }
162 
163     public static void setGlobalContext(Context ctx) {
164         globalContext = ctx;
165     }
166 
167     public static void setThreadContext(Context ctx) {
168         threadLocalContext.set(ctx);
169     }
170 }

```

Java EE Debug

```

1  package com.ebay.maui.controller;
2
3  import com.ebay.maui.DefaultSuite;
4  import com.ebay.maui.xmlsuite.XmlSuite;
5
6  public class DefaultSuite implements ISuite {
7
8      /**
9       * 
10      */
11
12      private static final long serialVersionUID = -1728730636766300291L;
13
14      private XmlSuite xmsuite;
15
16      public DefaultSuite() {
17          this.xmsuite = new DefaultXmlSuite();
18      }
19
20
21      public Object getAttribute(String name) {
22          // TODO Auto-generated method stub
23          return null;
24      }
25
26      public void setAttribute(String name, Object value) {
27          // TODO Auto-generated method stub
28      }
29
30      public Set<String> getAttributeNames() {
31          // TODO Auto-generated method stub
32          return null;
33      }
34
35      public Object removeattribute(String name) {
36          // TODO Auto-generated method stub
37          return null;
38      }
39
40      public String getName() {
41          return "Default suite";
42      }
43
44      public Map<String, ISuiteResult> getResults() {
45          // TODO Auto-generated method stub
46          return null;
47      }
48
49      public IObjectFactory getObjectFactory() {
50          // TODO Auto-generated method stub
51          return null;
52      }
53
54      public IObjectFactory2 getObjectFactory2() {
55          // TODO Auto-generated method stub
56          return null;
57      }
58
59      public String getOutputDirectory() {
60          // TODO Auto-generated method stub
61          return null;
62      }
63
64      public String getParallel() {
65          // TODO Auto-generated method stub
66          return null;
67      }
68
69      public String getParameter(String parameterName) {
70          // TODO Auto-generated method stub
71          return null;
72      }
73
74      public Map<String, Collection<ITestNGMethod>> getMethodsByGroups() {
75          // TODO Auto-generated method stub
76          return null;
77      }
78
79      public Collection<ITestNGMethod> getInvokedMethods() {
80          // TODO Auto-generated method stub
81          return null;
82      }
83
84      public List<ITestNGMethod> getAllInvokedMethods() {
85          // TODO Auto-generated method stub
86          return null;
87      }
88
89      public Collection<ITestNGMethod> getExcludedMethods() {
90          // TODO Auto-generated method stub
91          return null;
92      }
93
94      public void run() {
95          // TODO Auto-generated method stub
96      }
97
98      public String getHost() {
99          // TODO Auto-generated method stub
100         return null;
101     }
102
103     public SuiteRunstate getSuiteState() {
104         // TODO Auto-generated method stub
105         return null;
106     }
107
108     public IAnnotationFinder getAnnotationFinder() {
109         // TODO Auto-generated method stub
110         return null;
111     }
112
113     public XmlSuite getXmlSuite() {
114         // TODO Auto-generated method stub
115         return xmsuite;
116     }
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152

```

Read-Only Smart Insert 2 : 1

Java EE Debug

```

1  package com.ebay.maui.controller;
2
3  import com.ebay.maui.DefaultSuite;
4  import com.ebay.maui.xmlsuite.XmlSuite;
5
6  public class DefaultSuite implements ISuite {
7
8      /**
9       * 
10      */
11
12      private static final long serialVersionUID = -1728730636766300291L;
13
14      private XmlSuite xmsuite;
15
16      public DefaultSuite() {
17          this.xmsuite = new DefaultXmlSuite();
18      }
19
20
21      public Object getAttribute(String name) {
22          // TODO Auto-generated method stub
23          return null;
24      }
25
26      public void setAttribute(String name, Object value) {
27          // TODO Auto-generated method stub
28      }
29
30      public Set<String> getAttributeNames() {
31          // TODO Auto-generated method stub
32          return null;
33      }
34
35      public Object removeattribute(String name) {
36          // TODO Auto-generated method stub
37          return null;
38      }
39
40      public String getName() {
41          return "Default suite";
42      }
43
44      public Map<String, ISuiteResult> getResults() {
45          // TODO Auto-generated method stub
46          return null;
47      }
48
49      public IObjectFactory getObjectFactory() {
50          // TODO Auto-generated method stub
51          return null;
52      }
53
54      public IObjectFactory2 getObjectFactory2() {
55          // TODO Auto-generated method stub
56          return null;
57      }
58
59      public String getOutputDirectory() {
60          // TODO Auto-generated method stub
61          return null;
62      }
63
64      public String getParallel() {
65          // TODO Auto-generated method stub
66          return null;
67      }
68
69      public String getParameter(String parameterName) {
70          // TODO Auto-generated method stub
71          return null;
72      }
73
74      public Map<String, Collection<ITestNGMethod>> getMethodsByGroups() {
75          // TODO Auto-generated method stub
76          return null;
77      }
78
79      public Collection<ITestNGMethod> getInvokedMethods() {
80          // TODO Auto-generated method stub
81          return null;
82      }
83
84      public List<ITestNGMethod> getAllInvokedMethods() {
85          // TODO Auto-generated method stub
86          return null;
87      }
88
89      public Collection<ITestNGMethod> getExcludedMethods() {
90          // TODO Auto-generated method stub
91          return null;
92      }
93
94      public void run() {
95          // TODO Auto-generated method stub
96      }
97
98      public String getHost() {
99          // TODO Auto-generated method stub
100         return null;
101     }
102
103     public SuiteRunstate getSuiteState() {
104         // TODO Auto-generated method stub
105         return null;
106     }
107
108     public IAnnotationFinder getAnnotationFinder() {
109         // TODO Auto-generated method stub
110         return null;
111     }
112
113     public XmlSuite getXmlSuite() {
114         // TODO Auto-generated method stub
115         return xmsuite;
116     }
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152

```

Read-Only Smart Insert 2 : 1

```
121     return null;
122 }
123 }
124
125     public void run() {
126         // TODO Auto-generated method stub
127     }
128 }
129
130     public String getHost() {
131         // TODO Auto-generated method stub
132         return null;
133     }
134 }
135
136     public SuiteRunState getSuiteState() {
137         // TODO Auto-generated method stub
138         return null;
139     }
140 }
141
142     public IAnnotationFinder getAnnotationFinder() {
143         // TODO Auto-generated method stub
144         return null;
145     }
146 }
147
148     public XmSuite getXmSuite() {
149         // TODO Auto-generated method stub
150         return xmSuite;
151     }
152 }
153
154     public void addlistener(ITestNGlistener listener) {
155         // TODO Auto-generated method stub
156     }
157 }
158
159     public List<ITestNGMethod> getAllMethods() {
160         // TODO Auto-generated method stub
161         return null;
162     }
163 }
164
165     @Override
166     public Injector getParentInjector() {
167         // TODO Auto-generated method stub
168         return null;
169     }
170 }
171
172     @Override
173     public String getParentModule() {
174         // TODO Auto-generated method stub
175         return null;
176     }
177 }
178
179     @Override
180     public void setParentInjector(Injector org0) {
181         // TODO Auto-generated method stub
182     }
183 }
184 }
185 }
```

```
21     public class DefaultTestNGContext implements ITestContext {
22     /*
23     */
24
25     private static final long serialVersionUID = 3809658526152767666L;
26
27     ISuite suite;
28
29     public DefaultTestNGContext() {
30         this.suite = new DefaultSuite();
31     }
32
33     public Object getAttribute(String name) {
34         // TODO Auto-generated method stub
35         return null;
36     }
37
38     public void setAttribute(String name, Object value) {
39         // TODO Auto-generated method stub
40     }
41
42     public Set<String> getAttributeNames() {
43         // TODO Auto-generated method stub
44         return null;
45     }
46
47     public Object removeAttribute(String name) {
48         // TODO Auto-generated method stub
49         return null;
50     }
51
52     public String getName() {
53         // TODO Auto-generated method stub
54         return null;
55     }
56
57     public Date getstartDate() {
58         // TODO Auto-generated method stub
59         return null;
60     }
61
62     public Date getEndDate() {
63         // TODO Auto-generated method stub
64         return null;
65     }
66
67     public IResultMap getPassedTests() {
68         // TODO Auto-generated method stub
69         return null;
70     }
71
72     public IResultMap getSkippedTests() {
73         // TODO Auto-generated method stub
74         return null;
75     }
76
77     public IResultMap getFailedButWithinSuccessPercentageTests() {
78         // TODO Auto-generated method stub
79         return null;
80     }
81
82     public IResultMap getFailedTests() {
83         // TODO Auto-generated method stub
84         return null;
85 }
```

Java EE Debug

```

  87  public String[] getIncludedGroups() {
  88      // TODO Auto-generated method stub
  89      return null;
  90  }
  91 
  92  public String[] getExcludedGroups() {
  93      // TODO Auto-generated method stub
  94      return null;
  95  }
  96 
  97  public String getOutputDirectory() {
  98      try {
  99          return this.getIoms().getResource("/").getPath()
  100         + "/../../test-output/defaultSuite";
  101     } catch (Exception e) {
  102         return null;
  103     }
  104 }
  105 
  106 public ISuite getSuite() {
  107     return suite;
  108 }
  109 
  110 public ITestNGMethod[] getAllTestMethods() {
  111     // TODO Auto-generated method stub
  112     return null;
  113 }
  114 
  115 public String getHost() {
  116     // TODO Auto-generated method stub
  117     return null;
  118 }
  119 
  120 public Collection<ITestNGMethod> getExcludedMethods() {
  121     // TODO Auto-generated method stub
  122     return null;
  123 }
  124 
  125 public IResultMap getPassedConfigurations() {
  126     // TODO Auto-generated method stub
  127     return null;
  128 }
  129 
  130 public IResultMap getSkippedConfigurations() {
  131     // TODO Auto-generated method stub
  132     return null;
  133 }
  134 
  135 public IResultMap getFailedConfigurations() {
  136     // TODO Auto-generated method stub
  137     return null;
  138 }
  139 
  140 public XUnitTest getCurrentXmlTest() {
  141     // TODO Auto-generated method stub
  142     return null;
  143 }
  144 
  145 public List<Module> getGuiceModules(Class<? extends Module> cls) {
  146     // TODO Auto-generated method stub
  147     return null;
  148 }
  149 
  150 public void addGuiceModule(Class<? extends Module> cls, Module module) {
  151     // TODO Auto-generated method stub
  152 }
  153 
  154 public Injector getInjector(List<Module> moduleInstances) {
  155     // TODO Auto-generated method stub
  156     return null;
  157 }
  158 
  159 public void addInjector(List<Module> moduleInstances, Injector injector) {
  160     // TODO Auto-generated method stub
  161 }
  162 
  163 }
  164 }
  165 }
```

Read-Only Smart Insert 2 : 1

Java EE Debug

```

  102     return null;
  103 }
  104 
  105 public ISuite getSuite() {
  106     return suite;
  107 }
  108 
  109 public ITestNGMethod[] getAllTestMethods() {
  110     // TODO Auto-generated method stub
  111     return null;
  112 }
  113 
  114 public String getHost() {
  115     // TODO Auto-generated method stub
  116     return null;
  117 }
  118 
  119 public Collection<ITestNGMethod> getExcludedMethods() {
  120     // TODO Auto-generated method stub
  121     return null;
  122 }
  123 
  124 public IResultMap getPassedConfigurations() {
  125     // TODO Auto-generated method stub
  126     return null;
  127 }
  128 
  129 public IResultMap getSkippedConfigurations() {
  130     // TODO Auto-generated method stub
  131     return null;
  132 }
  133 
  134 public IResultMap getFailedConfigurations() {
  135     // TODO Auto-generated method stub
  136     return null;
  137 }
  138 
  139 public XUnitTest getCurrentXmlTest() {
  140     // TODO Auto-generated method stub
  141     return null;
  142 }
  143 
  144 public List<Module> getGuiceModules(Class<? extends Module> cls) {
  145     // TODO Auto-generated method stub
  146     return null;
  147 }
  148 
  149 public void addGuiceModule(Class<? extends Module> cls, Module module) {
  150     // TODO Auto-generated method stub
  151 }
  152 
  153 
  154 public Injector getInjector(List<Module> moduleInstances) {
  155     // TODO Auto-generated method stub
  156     return null;
  157 }
  158 
  159 public void addInjector(List<Module> moduleInstances, Injector injector) {
  160     // TODO Auto-generated method stub
  161 }
  162 
  163 }
  164 }
  165 }
```

Read-Only Smart Insert 2 : 1

```

1 package com.ebay.mauli.controller;
2 import org.junit.runner.RunWith;
3 import org.junit.runners.Suite;
4 /**
5  * @author junjshi, created on Aug 1, 2013
6  */
7
8 public class DefaultXmlSuite extends XmlSuite{
9
10    /**
11     *
12     */
13
14    private static final long serialVersionUID = -6658243628670353825L;
15
16 }

```

Markers Properties Servers Data Source Explorer Snippets Console Progress Search Results of running class RaiseClaimTest Package Explorer

No operations to display at this time.

Read-Only Smart Insert 1:1

```

19 public class EasyFilter {
20
21     ///////////////////// Operator Enumeration ///////////////////
22     static enum Operator {
23         Equals, EqualsIgnoreCase, Lt, Gt, Between, In, IsNull, Not, Contains, ContainsIgnoreCase, StartsWith, StartsWithIgnoreCase, EndsWith, EndsWithIgnoreCase, Or, And;
24     }
25
26     public static EasyFilter and(EasyFilter left, EasyFilter right) {
27         if(left==null)
28             throw new RuntimeException("EasyFilter passed as null; make sure that the left assignment is not null.");
29         if(right==null)
30             return new EasyFilter(left, right, Operator.And);
31         return left;
32     }
33
34     public static EasyFilter between(String name, Date value1, Date value2) {
35         return new EasyFilter(name, new Object[] { value1, value2 }, Operator.Between);
36     }
37
38     public static EasyFilter between(String name, Number value1, Number value2) {
39         return new EasyFilter(name, new Object[] { value1, value2 }, Operator.Between);
40     }
41
42     public static EasyFilter contains(String name, String value) {
43         return new EasyFilter(name, value, Operator.Contains);
44     }
45
46     public static EasyFilter containsIgnoreCase(String name, String value) {
47         return new EasyFilter(name, value, Operator.ContainsIgnoreCase);
48     }
49
50     public static EasyFilter endsWith(String name, String value) {
51         return new EasyFilter(name, value, Operator.EndsWith);
52     }
53
54     public static EasyFilter endsWithIgnoreCase(String name, String value) {
55         return new EasyFilter(name, value, Operator.EndsWithIgnoreCase);
56     }
57
58     public static EasyFilter equals(String name, Object value) {
59         return new EasyFilter(name, value, Operator.Equal);
60     }
61
62     public static EasyFilter equalsIgnoreCase(String name, String value) {
63         return new EasyFilter(name, value, Operator.EqualIgnoreCase);
64     }
65
66     public static EasyFilter gt(String name, Date value) {
67         return new EasyFilter(name, value, Operator.Gt);
68     }
69
70     public static EasyFilter gt(String name, Number value) {
71         return new EasyFilter(name, value, Operator.Gt);
72     }
73
74     public static EasyFilter gt(String name, Object[] values) {
75         return new EasyFilter(name, values, Operator.In);
76     }
77
78     public static EasyFilter isNull(String name) {
79         return new EasyFilter(name, (String[]) null, Operator.IsNull);
80     }
81
82 }

```

Read-Only Smart Insert 110 : 6

The screenshot shows the Java code for the `EasyFilter` class in an IDE. The code implements a filter mechanism based on various operators like `Between`, `Equal`, `NotEqual`, etc. It handles different data types including strings, numbers, and dates. The code includes annotations like `@Suppressions` and `@Override`. The IDE interface includes tabs for other files like `SpreadSheetUtil`, `CheckClaimTestP`, etc., and a toolbar with various icons.

```
94     public static EasyFilter lt(String name, Date value) {
95         return new EasyFilter(name, value, Operator.lt);
96     }
97
98     public static EasyFilter lt(String name, Number value) {
99         return new EasyFilter(name, value, Operator.lt);
100    }
101
102    @SuppressWarnings("deprecation")
103    public static void main(String[] st) {
104        EasyFilter f = EasyFilter.between("mydate", new Date("01-JAN-2008"),
105                                         new Date("01-JAN-2011"));
106
107        Map<String, Object> map = new HashMap<String, Object>();
108        map.put("mydate", new Date("01-JAN-2010"));
109        System.out.println(f.match(map));
110
111        map.put("mydate", new Date("01-JAN-2012"));
112        System.out.println(f.match(map));
113
114        // Date[] mydates = {new Date("01-JAN-2008"),new Date("01-JAN-2010"),new
115        // Date("01-JAN-2011")};
116
117        @SuppressWarnings("IncompleteSwitch")
118        private static boolean matchEasyFilter Filter,
119        Map<String, Object> parameters) {
120            String name = parameters.get(name) ? filter.name.toUpperCase() : null;
121            Object values[] = filter.values;
122            Operator operator = filter.operator;
123            EasyFilter left = filter.left;
124            EasyFilter right = filter.right;
125            //handle array, such as emolDriverTo.ADDRESS return a String[], so get the first value only
126            Object receiveValues = parameters.get(name);
127
128            if (operator.And.equals(operator)) {
129                return (left.match(parameters) && right.match(parameters));
130            } else if (operator.Or.equals(operator)) {
131                return (left.match(parameters) || right.match(parameters));
132            } else if (operator.Not.equals(operator)) {
133                return !right.match(parameters);
134            } else if (operator.IsNull.equals(operator)) {
135                return false;
136            } else if (operator.IsNotNull.equals(operator)) // Direct Null Check
137                return true;
138            && (receiveValues == null || (values.length == 1 && values[0] == null)) // Indirect Null
139            // Check
140            return (receiveValues == null);
141            } else if (operator.Equals.equals(operator)) {
142                if (receiveValues.getClass().isArray() && !values[0].getClass().isArray())
143                    receiveValues = ((Object[])receiveValues)[0];
144                return receiveValues.equals(values[0]);
145            } else if (operator.EqualsIgnoreCase.equals(operator)) {
146                if (receiveValues.getClass().isArray() && !values[0].getClass().isArray())
147                    receiveValues = ((Object[])receiveValues)[0];
148                return receiveValues.toString().toLowerCase()
149                .equals(values[0].toString().toLowerCase());
150            } else if (operator.In.equals(operator)) {
151                boolean found = false;
152
153                for (int i = 0; i < values.length; i++) {
154                    if (receiveValues.equals(values[i])) {
155                        found = true;
156                        break;
157                    }
158                }
159                return found;
160            } else if (values == null || values[0] == null) {
161                throw new RuntimeException(
162                    "Null values are not supported for Filter Operation: "
163                    + operator);
164            } else // By the time the control reaches here values is not null
165            // String specific: Contains, ContainsIgnoreCase
166            if (values[0] instanceof String) {
167                if (receiveValues.getClass().isArray())
168                    receiveValues = ((Object[])receiveValues)[0];
169                switch (operator) {
170                    case Contains:
171                        return receiveValues.toString()
172                            .contains(values[0].toString());
173                    case ContainsIgnoreCase:
174                        return receiveValues.toString().toLowerCase()
175                            .contains(values[0].toString().toLowerCase());
176                    case StartsWith:
177                        return receiveValues.toString()
178                            .startsWith(values[0].toString());
179                    case StartsWithIgnoreCase:
180                        return receiveValues.toString().toLowerCase()
181                            .startsWith(values[0].toString().toLowerCase());
182                    case EndsWith:
183                        return receiveValues.toString()
184                            .endsWith(values[0].toString());
185                    case EndsWithIgnoreCase:
186                        return receiveValues.toString().toLowerCase()
187                            .endsWith(values[0].toString().toLowerCase());
188                }
189            } else if (values[0] instanceof Number) // Number & Date specific:
190            // Between, Lt, Gt
191            if (receiveValues.getClass().isArmy())
192                receiveValues = ((Object[])receiveValues)[0];
193                BigDecimal val = new BigDecimal((receiveValues).toString());
194                BigDecimal leftValue = new BigDecimal(values[0].toString());
195                BigDecimal rightValue = null;
196
197                switch (operator) {
198                    case Between:
199                        rightValue = new BigDecimal(values[1].toString());
200                        return leftValue.compareTo(val) < 1
201                            && rightValue.compareTo(val) > -1;
202                    case Lt:
203                        return val.compareTo(leftValue) == -1;
204                    case Gt:
205                        return val.compareTo(leftValue) == 1;
206                    default:
207                        break;
208                }
209            } else if (values[0] instanceof Date) {
210                if (receiveValues.getClass().isArray())
211                    receiveValues = ((Object[])receiveValues)[0];
212                Date date = null;
213                +nu;
```

This screenshot shows the continuation of the `matchEasyFilter` method from the previous screenshot. It handles numeric and date comparisons, including `Between`, `Lt`, and `Gt` operators. The code uses `BigDecimal` for number comparisons and `Date` objects for date comparisons. The IDE interface remains consistent with the top screenshot.

```
145                rightValue = new BigDecimal(values[1].toString());
146                return leftValue.compareTo(val) < 1
147                    && rightValue.compareTo(val) > -1;
148            case Lt:
149                return val.compareTo(leftValue) == -1;
150            case Gt:
151                return val.compareTo(leftValue) == 1;
152            default:
153                break;
154            }
155        } else if (values[0] instanceof Date) {
156            if (receiveValues.getClass().isArray())
157                receiveValues = ((Object[])receiveValues)[0];
158            Date date = null;
159            +nu;
```

```

119     try {
120         date = DateFormat.getDateInstance().parse(
121             receiveValues.toString());
122     } catch (ParseException e) {
123         //throw new RuntimeException(e);
124         date = (Date)receiveValues;
125     }
126     Date dateLeft = (Date) values[0];
127     Date dateRight = null;
128
129     switch (operator) {
130     case Between:
131         dateRight = (Date) values[1];
132         return (dateLeft.before(date) || dateLeft.equals(date))
133             && (date.before(dateRight) || date
134                 .equals(dateRight));
135     case Lt:
136         return date.before(dateLeft);
137     case Gt:
138         return date.after(dateLeft);
139     default:
140         break;
141     }
142 }
143
144 throw new RuntimeException("Should not reach here. Not Implemented Yet"
145     + "n" + filter + "n" + parameters);
146 }
147
148 /////////////////////////////////////////////////////////////////// Static Methods ///////////////////////////////////////////////////////////////////
149
150 public static EasyFilter not(EasyFilter exp) {
151     return new EasyFilter(null, exp, Operator.Not);
152 }
153
154 public static EasyFilter or(EasyFilter left, EasyFilter right) {
155     return new EasyFilter(left, right, Operator.Or);
156 }
157
158 public static EasyFilter startsWith(String name, String value) {
159     return new EasyFilter(name, value, Operator.StartsWith);
160 }
161
162 public static EasyFilter startsWithIgnoreCase(String name, String value) {
163     return new EasyFilter(name, value, Operator.StartsWithIgnoreCase);
164 }
165
166 private String name;
167
168 private Object[] values;
169
170 private Operator operator;
171
172 private EasyFilter left;
173
174 private EasyFilter right;
175
176 private EasyFilter(EasyFilter left, EasyFilter right, Operator condition) {
177     this.left = left;
178     this.right = right;
179     this.operator = condition;
180 }
181
182 public EasyFilter(String name, Object value, Operator condition) {
183     this(name, new Object[] { value }, condition);
184 }
185
186 public EasyFilter(String name, Object[] values, Operator condition) {
187     super();
188     this.name = name;
189     this.values = values;
190     this.operator = condition;
191 }
192
193 public EasyFilter getLeft() {
194     return left;
195 }
196
197 public String getName() {
198     return name;
199 }
200
201 public Operator getOperator() {
202     return operator;
203 }
204
205 public EasyFilter getRight() {
206     return right;
207 }
208
209 public Object[] getValues() {
210     return values;
211 }
212
213 public boolean match(Map<String, Object> parameters) {
214     Map<String, Object> parameters2 = new HashMap<String, Object>();
215     for (Entry<String, Object> entry : parameters.entrySet()) {
216         parameters2.put(entry.getKey(), entry.getValue());
217     }
218     parameters2.put(entry.getKey(), entry.getValue());
219     return match(this, parameters2);
220 }
221
222 public void setLeft(EasyFilter left) {
223     this.left = left;
224 }
225
226 public void setName(String name) {
227     this.name = name;
228 }
229
230 public void setOperator(Operator condition) {
231     this.operator = condition;
232 }
233
234 public void setRight(EasyFilter right) {
235     this.right = right;
236 }
237
238 public void setValues(Object[] values) {
239     this.values = values;
240 }
241
242 public String toString() {
243     StringBuffer sb = new StringBuffer();
244     if (name != null)
245         sb.append(name + " " + operator.toString() + " "
246             + Arrays.toString(values));
247
248     return sb.toString();
249 }
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339

```

The screenshot shows an IDE interface with the title bar "Java EE" and "Debug". The code editor displays the `EasyFilter.class` file. The code is a Java class with various methods like `putEntry`, `match`, `setLeft`, `setName`, `setOperator`, `setRight`, `setValues`, `toString`, and `toXML`. The code uses reflection and string manipulation to build filter expressions.

```

310     parameters2.putEntry(entry.getKey().toUpperCase(), entry.getValue());
311   }
312   return match(this, parameters2);
313 }
314 
315 public void setLeft(EasyFilter left) {
316   this.left = left;
317 }
318 
319 public void setName(String name) {
320   this.name = name;
321 }
322 
323 public void setOperator(Operator condition) {
324   this.operator = condition;
325 }
326 
327 public void setRight(EasyFilter right) {
328   this.right = right;
329 }
330 
331 public void setValues(Object[] values) {
332   this.values = values;
333 }
334 
335 public String toString() {
336   StringBuffer sb = new StringBuffer();
337   if (name != null) {
338     sb.append(name + " " + operator.toString() + " "
339               + Arrays.toString(values));
340   } else {
341     sb.append(left != null ? left.toString() : "") + " "
342           + operator.toString() + " " + right.toString();
343   }
344 }
345 
346 return "(" + sb.toString() + ")";
347 }
348 
349 public String toXML() {
350   StringBuffer sb = new StringBuffer();
351 
352   if (left != null)
353     sb.append("<crit:left>").append(left.toXML())
354     .append("</crit:left>");
355   if (name != null)
356     sb.append("<crit:name>").append(name).append("</crit:name>");
357   if (operator != null)
358     sb.append("<crit:operator>").append(operator)
359     .append("</crit:operator>");
360   if (values != null)
361     sb.append("<crit:values>")
362       .forObject(obj : values)
363         .append("<val>").append(obj.toString())
364         .append("</val>")
365     .append("</crit:values>");
366 
367   if (right != null)
368     sb.append("<crit:right>").append(right.toXML())
369     .append("</crit:right>");
370 }
371 
372 }
373 }
374 
```

The screenshot shows an IDE interface with the title bar "Java EE" and "Debug". The code editor displays the `HTTPStatusCode.class` file. The code is a Java enum with over 60 entries, each mapping a numeric value to a status code and its description. The descriptions often include HTTP/1.1 status codes and their meanings.

```

2 /**
3 * This enumeration will capture all the HTTP error status codes 4XX and 5XX. For
4 * more details on HTTP status codes refer to
5 */
6 
7 /**
8 * @author open
9 */
10 
11 /**
12 * public enum HTTPStatusCode {
13 * 
14 * ///////////////////////////////////////////////////////////////////4xx (Request
15 * //////////////////////////////////////////////////////////////////
16 * // These status codes indicate that there was likely an error in the request
17 * // which prevented the server from being able to process it.
18 * @REQUEST(400), "Bad request",
19 * "The server didn't understand the syntax of the request.",
20 * // NOT_IMPLEMENTED(401), "Not authorized",
21 * // "The user is not authorized to make this request. The server might return this response for a page behind a login.",
22 * // FORBIDDEN(403),
23 * // "Directory listing denied, the server is refusing the request.",
24 * PAGE_NOT_FOUND(404), "Page not found",
25 * "The server can't find the requested page.", PAGE_MOVED_OR_NOT_AVAILABLEC
26 * "404, This page may have moved or is no longer available",
27 * "The server can't find the requested page.", PROBLEM_LOADING_PAGE
28 * "404, The server can't find the requested page.", PROBLEM_LOADING_PAGE
29 * // METHOD_NOT_ALLOWED(405), "Method not allowed",
30 * // "The method specified in the request is not allowed.",
31 * // NOT_ACCEPTABLE(406), "Not acceptable",
32 * // "The server can't accept the entity characteristics requested",
33 * // PROXY_AUTHENTICATION_REQUIRED(407), "Proxy authentication required",
34 * // "The requestor has to authenticate using a proxy.",
35 * REQUEST_TIMEOUT(408), "Request timeout",
36 * "The server is taking too long waiting for the request",
37 * // CONFLICT(409), "Conflict",
38 * // "The server encountered a conflict fulfilling the request."
39 * // GONE(410), "Gone"
40 * // "The server returns this response when the requested resource has been permanently removed. It is similar to a 404 Not Found."
41 * LENGTH_REQUIRED(411), "Length required",
42 * // "The server won't accept the request without a valid Content-length header field.",
43 * PRECONDITION_FAILED(412), "Precondition failed",
44 * // "The server can't meet one of the conditions that the requestor put on the request.",
45 * REQUEST_ENTITY_TOO_LARGE(413), "Request entity too large",
46 * // "The server can't process the request because it is too large for the server to handle."
47 * // REQUEST_URI_TOO_LONG(414), "Request URI is too long",
48 * // "The requested URL typically, a URL is too long for the server to process."
49 * // UNSUPPORTED_MEDIA_TYPE(415), "Unsupported media type",
50 * // "The request is in a format not support by the requested page",
51 * // REQUESTED_RANGE_NOT_SATISFIABLE(416), "Requested range not satisfiable",
52 * // "The server returns this status code if the request is for a range not available for the page."
53 * EXPECTATION_FAILED(417),
54 * // "Expectation failed", "The server can't meet the requirements of the Expect request-header field."
55 * 
56 * ///////////////////////////////////////////////////////////////////5xx (Server error)
57 * //////////////////////////////////////////////////////////////////
58 * // These status codes indicate that the server had an internal error when
59 * // trying to process the request.
60 * // These errors tend to be with the server itself, not with the request.
61 * INTERNAL_SERVER_ERROR(500), "Internal server error",
62 * "The server encountered an error and can't fulfill the request."
63 * // NOT_IMPLEMENTED(501), "Not implemented",
64 * // "The server doesn't have the functionality to fulfill the request. For instance, the server might return this code when it doesn't recognize the request method."
65 * BAD_GATEWAY
66 * 
```

```

HTTPStatusCode.java
=====
58     // These status codes indicate that the server had an internal error when
59     // trying to process the request.
60     // The server tends to be with the server itself, not with the request.
61     INTERNAL_SERVER_ERROR<60>, "Internal Server Error",
62         *The server encountered an error and can't fulfill the request.*,
63     // NOT_IMPLEMENTED<61>, "Not Implemented",
64     // "The server doesn't have the functionality to fulfill the request. For instance, the server might return this code when it doesn't recognize the request method.*,
65     BAD_GATEWAY<62>,
66         *Bad gateway*,
67         *The server was acting as a gateway or proxy and received an invalid response from the upstream server.*,
68     SERVICE_UNAVAILABLE<63>,
69         *Service Unavailable*,
70         *The service is currently unavailable because it is overloaded or down for maintenance.*,
71     FEATURE_UNAVAILABLE<64>,
72         *Feature Unavailable*,
73         *The server is currently unavailable because it is overloaded or down for maintenance.*,
74     GATEWAY_TIMEOUT<65>,
75         *Gateway Timeout*,
76         *The server was acting as a gateway or proxy and didn't receive a timely request from the upstream server.*,
77     HTTP_VERSION_NOT_SUPPORTED<66>,
78         *HTTP version not supported*,
79     PAGE_NOT_RESPONDING<67>,
80         *The server doesn't support the HTTP protocol version used in the request.*,
81     TECHNICAL_ERROR<68>,
82         *Technical Error*,
83         *Technical Error*;
84
85     public static HTTPStatusCode getByCode(int code) {
86         for (HTTPStatusCode status : values()) {
87             if (status.getCode() == code)
88                 return status;
89         }
90         throw new RuntimeException("HTTPStatusCode: " + code
91             + " not supported.");
92     }
93     ///////////////////////////////////////////////////////////////////member variables, getters, setters and constructors
94     private int code;
95     private String title;
96     private String description;
97
98     private HTTPStatusCode(int code, String title, String description) {
99         this.code = code;
100        this.title = title;
101        this.description = description;
102    }
103
104    public int getCode() {
105        return code;
106    }
107
108    public String getDescription() {
109        return description;
110    }
111
112    public String getTitle() {
113        return title;
114    }
115
116    public String toString() {
117        return name() + "(" + getCode() + ") - " + getTitle() + " [" +
118            + getDescription() + "]";
119    }
120}
121
122

```

```

Project Explorer
=====
IContextAttribut
IConverter.class
Keyword.class
KeywordProvider
Logging.class
SoftAssertion.c
TearDownService
TestParameter.c
HTTPStatusCode
IContextAttributeListener
com.ebay.maul.controller
AbstractPageListener.class
AnnotationTransformer.class
Assertion.class
Context.class
ContextManager.class
DefaultSuite.class
DefaultTestNGContext.class
DefaultXmSuite.class
EasyFilter.class
HTTPStatusCode.class
IContextAttributeListener.class
IConverter.class
Keyword.class
KeywordProvider.class
Logging.class
SoftAssertion.class
TearDownService.class
TestParameter.class
TestPlan.class
TestRetryAnalyzer.class
TestRetryListener.class
com.ebay.maul.driver.api
com.ebay.maul.driver.db
com.ebay.maul.driver.email
com.ebay.maul.driver.ssh
com.ebay.maul.driver.web
com.ebay.maul.driver.web.element
com.ebay.maul.driver.web.factory
com.ebay.maul.exception
com.ebay.maul.helper
com.ebay.maul.reporter
com.ebay.maul.reporter.pluginmodel
reporter.cs
reporter.images.lightbox
reporter.images.mktree
reporter.images.yukontoolbox
reporter.js
templates
META-INF
tmp
guice-3.0.jar
jaxva.inject-1.jar

```

```

package com.ebay.maul.controller;
public interface IConverter {
    public String convertURL(String url) throws Exception;
}

```

No operations to display at this time.

```

private static Map<String, KeywordProvider> s_resourceBaseNameProviderMap = new Hashtable<String, KeywordProvider>();
static {
    s_resourceBaseNameProviderMap.put(KeywordProvider.PropertiesKeywordProvider.DEFAULT_RESOURCE_BASE_NAME, new KeywordProvider.PropertiesKeywordProvider());
}

private static KeywordProvider getProviderByResourceBaseName(String resourceBaseName) {
    KeywordProvider provider = s_resourceBaseNameProviderMap.get(resourceBaseName.toLowerCase());
    if (null == provider) {
        provider = new KeywordProvider.PropertiesKeywordProvider(resourceBaseName);
        s_resourceBaseNameProviderMap.put(resourceBaseName.toLowerCase(), provider);
    }
    return provider;
}

private static String _getValue(String input, Object[] values) {
    if (null == values)
        return input;
    Matcher matcher = Pattern.compile("\\$\\{\\d+\\}\\$").matcher(input);
    String switch_ = input;
    String groupValue = null;
    Object tempValue = null;
    while (true) {
        groupValue = matcher.group();
        int index = Integer.parseInt(groupValue.replaceAll("\\{\\}", ""));
        if (index >= 0 && index < values.length) {
            tempValue = values[index];
            switch_ = switch_.replace(groupValue, (null == tempValue) ? "?NULL?" : tempValue.toString());
        }
    }
    return switch_;
}

public static String getClass(> callingClass, String key) {
    KeywordProvider provider = s_resourceBaseNameProviderMap.get(KeywordProvider.PropertiesKeywordProvider.DEFAULT_RESOURCE_BASE_NAME);
    return provider.getKeyword(callingClass, key);
}

public static String getClass(> callingClass, String key, Object[] values) {
    KeywordProvider provider = s_resourceBaseNameProviderMap.get(KeywordProvider.PropertiesKeywordProvider.DEFAULT_RESOURCE_BASE_NAME);
    String originalValue = provider.getKeyword(site, callingClass, key);
    return _getValue(originalValue, values);
}

public static String getClass(> callingClass, String key, String site, Object[] values) {
    KeywordProvider provider = s_resourceBaseNameProviderMap.get(KeywordProvider.PropertiesKeywordProvider.DEFAULT_RESOURCE_BASE_NAME);
    String originalValue = provider.getKeyword(site, callingClass, key);
    return _getValue(originalValue, values);
}

public static String getKey(String key) {
    KeywordProvider provider = s_resourceBaseNameProviderMap.get(KeywordProvider.PropertiesKeywordProvider.DEFAULT_RESOURCE_BASE_NAME);
    Class<?> callingClass = ClassHelper.getCallerEncloser();
    return provider.getKeyword(callingClass, key);
}

```

```

 78  public static String get(String key, Object[] values) {
 79      KeywordProvider provider = s_resourceBaseNameProviderMap.get(KeywordProvider.PropertiesKeywordProvider.DEFAULT_RESOURCE_BASE_NAME);
 80      Class callingClass = ClassContextHelper.getCallerEncloser();
 81      String originalValue = provider.getKeyword(callingClass, key);
 82      return provider.getKeyword(site, callingClass, key);
 83  }
 84
 85  public static String get(String key, String site) {
 86      KeywordProvider provider = s_resourceBaseNameProviderMap.get(KeywordProvider.PropertiesKeywordProvider.DEFAULT_RESOURCE_BASE_NAME);
 87      Class callingClass = ClassContextHelper.getCallerEncloser();
 88      return provider.getKeyword(site, callingClass, key);
 89  }
 90
 91  public static String get(String key, String site, Object[] values) {
 92      KeywordProvider provider = s_resourceBaseNameProviderMap.get(KeywordProvider.PropertiesKeywordProvider.DEFAULT_RESOURCE_BASE_NAME);
 93      Class callingClass = ClassContextHelper.getCallerEncloser();
 94      String originalValue = provider.getKeyword(callingClass, key);
 95      return provider.getKeyword(site, callingClass, key);
 96  }
 97
 98  public static String getByResourceBaseName(Class<?> callingClass, String resourceBaseName, String key) {
 99      KeywordProvider provider = s_getProviderMapByResourceBaseName(resourceBaseName);
100      return provider.getKeyword(callingClass, key);
101  }
102
103  public static String getByResourceBaseName(Class<?> callingClass, String resourceBaseName, String key, Object[] values) {
104      KeywordProvider provider = s_getProviderMapByResourceBaseName(resourceBaseName);
105      String originalValue = provider.getKeyword(callingClass, key);
106      return provider.getValue(originalValue, values);
107  }
108
109  public static String getByResourceBaseName(Class<?> callingClass, String resourceBaseName, String key, String site) {
110      KeywordProvider provider = s_getProviderMapByResourceBaseName(resourceBaseName);
111      return provider.getKeyword(site, callingClass, key);
112  }
113
114  public static String getByResourceBaseName(Class<?> callingClass, String resourceBaseName, String key, String site, Object[] values) {
115      KeywordProvider provider = s_getProviderMapByResourceBaseName(resourceBaseName);
116      String originalValue = provider.getKeyword(site, callingClass, key);
117      return provider.getValue(originalValue, values);
118  }
119
120  public static String getByResourceBaseName(String resourceBaseName, String key) {
121      KeywordProvider provider = s_getProviderMapByResourceBaseName(resourceBaseName);
122      Class callingClass = ClassContextHelper.getCallerEncloser();
123      String originalValue = provider.getKeyword(callingClass, key);
124      return provider.getValue(originalValue, values);
125  }
126
127  public static String getByResourceBaseName(String resourceBaseName, String key, Object[] values) {
128      KeywordProvider provider = s_getProviderMapByResourceBaseName(resourceBaseName);
129      Class callingClass = ClassContextHelper.getCallerEncloser();
130      String originalValue = provider.getKeyword(callingClass, key);
131      return provider.getValue(originalValue, values);
132  }
133
134  public static String getByResourceBaseName(String resourceBaseName, String key, String site) {
135      KeywordProvider provider = s_getProviderMapByResourceBaseName(resourceBaseName);
136      Class callingClass = ClassContextHelper.getCallerEncloser();
137      return provider.getKeyword(site, callingClass, key);
138  }
139
140  public static String getByResourceBaseName(String resourceBaseName, String key, String site, Object[] values) {
141      KeywordProvider provider = s_getProviderMapByResourceBaseName(resourceBaseName);
142      Class callingClass = ClassContextHelper.getCallerEncloser();
143      return provider.getKeyword(site, callingClass, key);
144  }
145
146  }
147}

```

```
14
15  @SupressWarnings("rawtypes")
16  public interface KeywordProvider {
```

```
17
18  public final class PropertiesKeywordProvider implements KeywordProvider {
19      private static final HashSet<String> _loadedKeywordsPackageSet = new HashSet<String>();
20      private static final String _localizedString = new HashMap<String, String>();
21      private static final String DEFAULT_KEYWORD_KEY_PREFIX = "default";
22      private String _resourceBaseName = null;
23
24      public static final String DEFAULT_RESOURCE_BASE_NAME = "keywords";
25
26      public PropertiesKeywordProvider() {
27          this._resourceBaseName = DEFAULT_RESOURCE_BASE_NAME;
28      }
29
30      public PropertiesKeywordProvider(String resourceBaseName) {
31          this._resourceBaseName = (null == resourceBaseName || "".equals(resourceBaseName)) ? DEFAULT_RESOURCE_BASE_NAME : resourceBaseName;
32      }
33
34      /**
35      * Load keywords for a domain flow.
36      *
37      * @param flowless
38      * @throws Exception
39      */
40
41      private void _loadKeywords(String site, Class stepClass) {
42          synchronized (PropertiesKeywordProvider.class) {
43              // to load default properties
44              String packagePath = stepClass.getPackageName();
45
46              if (_isLoadedKeywordsPackageSet.contains(_resourceBaseName + "." + DEFAULT_KEYWORD_KEY_PREFIX + "." + packagePath + ".keywords")) {
47                  ResourceBundle defaultBundle = _pickResourceBundle(stepClass);
48                  if (null != defaultBundle)
49                      logger.warn("WARN: resource '" + packagePath + "." + _resourceBaseName + ".properties" + " not found.");
50                  else
51                      Enumeration<String> keys = defaultBundle.getKeys();
52
53                      String identifier = null;
54                      String value = null;
55                      while (keys.hasMoreElements()) {
56                          identifier = keys.nextElement();
57                          value = defaultBundle.getString(identifier);
58
59                          if (!identifier.startsWith(packagePath))
60                              identifier = packagePath + "." + identifier;
61
62                          s_keywordsMap.put(_resourceBaseName + "." + DEFAULT_KEYWORD_KEY_PREFIX + "." + identifier, value);
63                      }
64
65                      _loadedKeywordsPackageSet.add(_resourceBaseName + "." + DEFAULT_KEYWORD_KEY_PREFIX + "." + packagePath + ".keywords");
66
67                  // to load the particular site properties
68                  ResourceBundle localizationBundle = _pickResourceBundle(site);
69                  if (null == localizationBundle)
70                      logger.warn("WARN: resource '" + packagePath + "." + _resourceBaseName + "." + site.toLowerCase() + ".properties" + " not found.");
71                  else {
72                      Enumeration<String> keys = localizationBundle.getKeys();
73                      String identifier = null;
74                      String value = null;
75                      while (keys.hasMoreElements()) {
76                          identifier = keys.nextElement();
77                          value = localizationBundle.getString(identifier);
78                          if (site.equalsIgnoreCase("RU") || site.equalsIgnoreCase("HK")) {
79                              try {
80                                  value = new String(value.getBytes("ISO8859-1"), "UTF-8");
81                              } catch (UnsupportedEncodingException e) {
82                                  }
83
84                          if (identifier.startsWith(packagePath))
85                              identifier = packagePath + "." + identifier;
86
87                          s_keywordsMap.put(_resourceBaseName + "." + site + "." + identifier, value);
88                      }
89
90                      _loadedKeywordsPackageSet.add(_resourceBaseName + "." + site + "." + packagePath + ".keywords");
91                  }
92
93
94                  private ResourceBundle _pickResourceBundle(Class cls) {
95                      String packagePath = cls.getPackageName();
96                      try {
97                          return ResourceBundle.getBundle(packagePath + "." + _resourceBaseName, new Locale("", "", ""));
98                      } catch (MissingResourceException e) {
99                          return null;
100                      }
101
102
103                  private ResourceBundle _pickResourceBundle(Class cls, String site) {
104                      String packagePath = cls.getPackageName();
105                      try {
106                          return ResourceBundle.getBundle(packagePath + "." + _resourceBaseName + ((null == site || "".equals(site)) ? "" : "." + site.toLowerCase()));
107                      } catch (MissingResourceException e) {
108                          return null;
109                      }
110
111                  private String getLocalizedSite() {
112                      String site = null;
113                      if (ContextManager.getThreadContext().getGbhSite() != null) {
114                          site = ContextManager.getThreadContext().getGbhSite();
115                      } else {
116                          site = ContextManager.getThreadContext().getSite();
117                      }
118
119                      return site;
120
121
122                  public String getKeyword(Class stepClass, String identifier) {
123                      return getKeyword(getLocalizedSite(), stepClass, identifier);
124
125
126                  public String getKeyword(String site, Class stepClass, String identifier) {
127                      if (_isLoadedKeywordsPackageSet.contains(_resourceBaseName + "." + site + "." + stepClass.getPackageName() + ".keywords"))
128                          _loadKeywords(site, stepClass);
129
130
131                      boolean localizedBuild = Boolean.parseBoolean((String) ContextManager.getGlobalContext().getAttribute(Context.LOCALIZED_BUILD));
132
133                      if (localizedBuild) // false to skip the site related picking
134                          if (s_keywordsMap.containsKey(_resourceBaseName + "." + site + "." + stepClass.getCanonicalName() + "." + identifier))
135                              return s_keywordsMap.get(_resourceBaseName + "." + site + "." + stepClass.getCanonicalName() + "." + identifier);
136
137                      if (s_keywordsMap.containsKey(_resourceBaseName + "." + DEFAULT_KEYWORD_KEY_PREFIX + "." + stepClass.getCanonicalName() + "." + identifier))
138                          return s_keywordsMap.get(_resourceBaseName + "." + DEFAULT_KEYWORD_KEY_PREFIX + "." + stepClass.getCanonicalName() + "." + identifier);
139
140
141                      // throw new
142                      // RuntimeException("No mappings found in the resource(base-name: " +
```

The screenshot shows the Java code for the `KeywordProvider` class. The code handles resource bundle retrieval, localized site identification, keyword extraction from step classes, and resource base name management. It includes exception handling for missing resources and uses context manager threads to get site names.

```
97     return ResourceBundle.getBundle(packagePath + "." + m_resourceBaseName, new Locale("", "", ""));  
98   } catch (MissingResourceException e) {  
99     return null;  
100   }  
101 }  
102 private ResourceBundle _pickResourceBundle(Class clazz, String site) {  
103   String packagePath = clazz.getPackageName();  
104   try {  
105     return ResourceBundle.getBundle(packagePath + "." + m_resourceBaseName + ((null == site || "".equals(site)) ? "" : "_" + site.toLowerCase()));  
106   } catch (MissingResourceException e) {  
107     return null;  
108   }  
109 }  
110 private String getLocalizedSite() {  
111   String site = null;  
112   if (ContextManager.getThreadContext().getGlobeSite() != null) {  
113     site = ContextManager.getThreadContext().getGlobeSite();  
114   } else {  
115     site = ContextManager.getThreadContext().getSite();  
116   }  
117   return site;  
118 }  
119 public String getKeyword(CClass stepClass, String identifier) {  
120   return getKeyWord(getLocalizedSite(), stepClass, identifier);  
121 }  
122 public String getKeyword(String site, Class stepClass, String identifier) {  
123   if (s_loadKeywords.keySet(m_resourceBaseName + "." + site + "." + stepClass.getPackageName() + ".keywords")) {  
124     _loadKeywords(site, stepClass);  
125   }  
126   boolean localizedBuild = Boolean.parseBoolean(ContextManager.getGlobalContext().getAttribute(Context.LOCALIZED_BUILD));  
127   if (LocalizedBuild) // false to skip the site related picking  
128     if (s_keywordsMap.containsKey(m_resourceBaseName + "." + site + "." + stepClass.getCanonicalName() + "_" + identifier))  
129       return s_keywordsMap.get(m_resourceBaseName + "." + site + "." + stepClass.getCanonicalName() + "_" + identifier);  
130   }  
131   if (s_keywordsMap.containsKey(m_resourceBaseName + "," + DEFAULT_KEYWORD_KEY_PREFIX + "," + stepClass.getCanonicalName() + "," + identifier))  
132     return s_keywordsMap.get(m_resourceBaseName + "," + DEFAULT_KEYWORD_KEY_PREFIX + "," + stepClass.getCanonicalName() + "," + identifier);  
133   }  
134   throw new RuntimeException("No mappings found in the resource(base-name: " +  
135     m_resourceBaseName + ") for keyword: " +  
136     stepClass.getCanonicalName() +  
137     "," + identifier + " and site=" + site);  
138 }  
139 public String getResourceBaseName() {  
140   return m_resourceBaseName;  
141 }  
142 public Logger logger = Logging.getLogger(KeywordProvider.class);  
143 public String getKeyWord(Class stepClass, String identifier);  
144 public String getKeyword(String site, CClass stepClass, String identifier);  
145 }
```

The screenshot shows the `Logging` class code. It manages a map of loggers, registers screenshot listeners, and logs screenshots. It includes error handling for screenshot listener exceptions and provides static methods for logging errors and creating root loggers.

```
47   public class Logging {  
48     private static Map<String, Map<String, Map<String, List<String>>> pageListenerMap = Collections.  
49     @SynchronizedMap<new HashMap<String, Map<String, Map<String, List<String>>>>();  
50     private static final Set<IScreenshotListener> screenshotListeners = new LinkedHashSet<IScreenshotListener>;  
51  
52     /**  
53      * Ability to register IScreenshotListeners.  
54      */  
55     public static void registerScreenshotListener(  
56       IScreenshotListener screenshotListener) {  
57       screenshotListeners.add(screenshotListener);  
58     }  
59  
60     public static void logScreenshot(final ScreenShot screenshot) {  
61       for (final IScreenshotListener screenshotListener : screenshotListeners) {  
62         new Thread() {  
63           public void run() {  
64             try {  
65               screenshotListener.onScreenshotCaptured(screenshot);  
66             } catch (Exception e) {  
67               // catch all listener impl exceptions to keep  
68               // continue with tests execution.  
69               // who ever implementing listener they must handle the  
70               // exceptions properly.  
71               System.err.  
72                 .println("Error in ScreenshotListener implementation "  
73                   + screenshotListener.getClass()  
74                   + " -"  
75                   + e.getName());  
76               + e.getMessage());  
77               // stacktrace will help to pin point issue.  
78               e.printStackTrace();  
79             }  
80           }  
81         }.start();  
82       }  
83     }  
84  
85     /**  
86      * Log method  
87      * @param message  
88      */  
89     public static void error(String message) {  
90       message = "<!--><font color='#FF0000'>" + message  
91       + "</font><br></li>";  
92       log(null, message, false, false);  
93     }  
94  
95     public static Logger getRootLogger(CClass c) {  
96       booted...  
97       .hasMoreElements());  
98       if (!rootIsConfigured) {  
99         BasicConfigurator.configure();  
100        Logger.root.setLevel(Level.INFO);  
101        Appender appender = (Appender) Logger.getLogger()  
102          .getAllAppenders().nextElement();  
103        appender.setLayout(new PatternLayout("%-5p %d [%t]: %m%n"));  
104      }  
105      return Logger.getLogger(c);  
106    }  
107  
108    public static void checkLog(Logger logger, String className, String errorMessage) {  
109    }  
110  }
```

The screenshot shows an IDE interface with two tabs open. The top tab displays the source code for the `Logging` class. The code is annotated with various markers such as `Map-String`, `Map-String, Map-String`, `List<String>>`, `Map<String>`, `MethodSignature`, `TestResult`, and `StringHelper`. The code implements methods for logging different types of messages, including `info`, `log`, and `logToStandardOutput`, and handles `PageListener` logs.

```
111 public static Map<String, Map<String, Map<String, List<String>>> getPageListenerLog() {
112     String pageListenerClassName;
113     return pageListenerLogMap.get(pageListenerClassName);
114 }
115
116 public static List<String> getPageListenerLogByMethodInfo(
117     ITestResult testResult) {
118
119     for (Entry<String, Map<String, Map<String, List<String>>> listenerEntry : pageListenerLogMap
120         .entrySet()) {
121         if (!PageUtil.getInstance().isTestResultAffected(
122             listenerEntry.getKey()))
123             continue;
124
125         Map<String, Map<String, List<String>> pageMap = listenerEntry
126             .getValue();
127         for (Entry<String, Map<String, List<String>> pageEntry : pageMap
128             .entrySet()) {
129             Map<String, List<String>> errorMap = pageEntry.getValue();
130             String methodInstanceId = StringHelper.constructMethodSignature(
131                 testResult.getMethod().getConstructorOrMethod()
132                     .getMethod(), testResult.getParameters());
133             return errorMap.get(methodInstanceId);
134         }
135     }
136
137     return null;
138 }
139
140 /**
141 * Log method
142 *
143 * @param message
144 */
145 public static void info(String message) {
146     message = "<li><font color="#008000">" + message + "</font></li>";
147     log(null, message, false, false);
148 }
149
150 /**
151 * Log method
152 *
153 * @param message
154 */
155 public static void log(String message, boolean logToStandardOutput) {
156     log(null, message, false, logToStandardOutput);
157 }
158
159 /**
160 * Log
161 *
162 * @param message
163 * @param logToStandardOutput
164 */
165 public static void log(String url, String message, boolean failed,
166     boolean logToStandardOutput) {
167
168     if (message == null)
169         message = "";
170
171     message = message.replaceAll("\n", "<br/>");
172 }
```

The screenshot shows an IDE interface with two tabs open. The bottom tab displays the source code for the `Logging` class. The code is annotated with `String`, `SimpleDateFormat`, `MethodSignature`, `TestResult`, and `StringHelper`. The code implements methods for logging API calls, mail steps, REST API calls, and SSH calls, including handling of failed operations.

```
174     message = message.replaceAll("\n", "<br/>");
175
176     if (failed) {
177         message = "<span style='font-weight:bold;color:#FF0066;'>" +
178             message + "</span>";
179     }
180
181     Reporter.log(message, logToStandardOutput);
182     //Reporter.log(escape(message), logToStandardOutput); // Fix for testing6.
183 }
184
185 /**
186 * public static String escape(String message) {
187     return message.replaceAll("<br>", "&bramp;").replaceAll("< ", "&#01100");
188     .replaceAll("<*>", "&gtamp;*&lt;");
189 }
190
191 /**
192 * public static String unescape(String message) {
193     message = message.replaceAll("<bramp;/>", "\n").replaceAll("&#01100", " ");
194     .replaceAll("&gtamp;*&lt;", "<*>");
```

```

239
240     public static void logWebStep(String url, String message, boolean failed) {
241         logUrl(url, getDate(), "Output: " + message + "<br>", failed, false);
242     }
243
244     public static void logWebStep(String url, String message, boolean failed) {
245         logUrl(url, getDate() + (failed ? "↳ Failed Step</b>" : getDate())
246                 + " Step") + message + "</li>", failed, false);
247     }
248
249     public static String buildScreenshotString(Screenshot screenshot) {
250         StringBuffer sbMessage = new StringBuffer("<br>");
251         if (screenshot != null) {
252             sbMessage.append("a href=\"" + screenshot.getLocation()
253                         + " target=_blank>" + screenshot.getLocation());
254             if (screenshot.getCallLog() != null)
255                 sbMessage.append("a href=\"" + screenshot.getCallLog()
256                             + " target=_blank>" + screenshot.getCallLog());
257             if (screenshot.getImagePath() != null)
258                 sbMessage.append("a href=\"" + screenshot.getImagePath()
259                             + " target=_blank>" + screenshot.getImagePath());
260             if (screenshot.getHTMLSourcePath() != null)
261                 sbMessage.append("a href=\"" + screenshot.getHTMLSourcePath()
262                             + " target=_blank>" + screenshot.getHTMLSourcePath());
263         }
264         if (screenshot.getImagePath() != null)
265             sbMessage.append("img alt=" + screenshot.getImagePath());
266         if (screenshot.getCallLog() != null)
267             sbMessage.append("img alt=" + screenshot.getCallLog());
268         if (screenshot.getPageId() != null)
269             sbMessage.append("a href=\"" + screenshot.getPageId()
270                             + " target=_blank>" + screenshot.getPageId());
271         return sbMessage.toString();
272     }
273
274     /**
275      * Log method
276      *
277      * @param message
278      */
279     public static void warning(String message) {
280         message = "<li>font color:#FF0000>" + message + "</font></li>";
281         logNull(message, false, false);
282     }
283
284     /**
285      * Log API steps with url, request, response, headers
286      *
287      * @param endPoint
288      * @param requestHeaders
289      * @param requestXML
290      * @param responseHeaders
291      * @param responseXML
292      * @return
293     */
294     public static String logAPIStep(String endPoint, Properties regHeaders,
295         String requestXML, Properties resHeaders, String responseXML) {
296         return logAPIStep(endPoint, regHeaders, requestXML, resHeaders,
297             responseXML, null, null);
298     }
299
300     /**
301      * Log API steps with url, request, response, headers,golden file
302      *
303      * @param endPoint
304      * @param requestHeaders
305      * @param requestXML
306      * @param responseHeaders
307      * @param responseXML
308      * @param goldenFile
309      * @param onlCompareConfig
310      * @return
311     */
312     public static String logAPIStep(String endPoint, Properties regHeaders,
313         String requestXML, Properties resHeaders, String responseXML,
314         String inputFile, Boolean onlCompareConfig) {
315         if (ContextManager.getGlobalContext() == null)
316             || ContextManager.getGlobalContext().getTestNGContext() == null)
317             return "";
318         StringBuffer sbMessage = new StringBuffer("<br>");
319         sbMessage.append("a href=\"" + endPoint + " target=_blank>url</a>");
320
321         String inputfilename = URLHelper.getandomtaskCode("son-input");
322         String outputfilename = URLHelper.getandomtaskCode("son-output");
323         String subFolder = ContextManager.getGlobalContext().getTestNGContext()
324             .getSuite().getSuite();
325         String outputDir = ContextManager.getGlobalContext()
326             .getOutputDirectory();
327
328         if (regHeaders != null) {
329             FileWriter fw = null;
330             try {
331                 fw = new FileWriter(outputDir + "/xmls/" + inputfilename
332                     + ".txt");
333                 fw.write("Content-Type: " + Headers);
334                 for (Entry<Object, Object> entry : regHeaders.entrySet()) {
335                     fw.write(entry.getKey() + "=" + entry.getValue() + "\n");
336                 }
337                 sbMessage.append("a href=\"" + subFolder + "/xmls/"
338                     + inputfilename
339                     + ".txt" + " target=_blank>request headers</a>");
340             } catch (IOException e) {
341                 finally {
342                     if (fw != null)
343                         try {
344                             fw.close();
345                         } catch (IOException e) {
346                         e.printStackTrace();
347                     }
348                 }
349             }
350         }
351         if (requestXML != null) {
352             try {
353                 FileHelper.writeToFile(outputDir + "/xmls/" + inputfilename
354                     + ".xml", requestXML);
355             } catch (IOException e) {
356                 e.printStackTrace();
357             }
358             sbMessage.append("a href=\"" + subFolder + "/xmls/"
359                     + inputfilename + ".xml" + " target=_blank>input</a>");
360         }
361         if (resHeaders != null) {
362
363

```

```

362
363     if (reqHeaders != null) {
364         try {
365             FileWriter fw = null;
366             fw = new FileWriter(outputDir + "/xmls/" + outputfilename
367                                 + ".txt");
368             fw.write("#Response Headers\n");
369             for (Entry<Object, Object> entry : reqHeaders.entrySet()) {
370                 fw.write(entry.getKey() + ":" + entry.getValue() + "\n");
371             }
372             sbMessage.append("<a href=\"" + subFolder + "/xmls/"
373                             + outputfilename
374                             + ".txt" + "\" target=_responseheaders-response headers=></a>");
375         } catch (IOException e) {
376             finally {
377                 if (fw != null) {
378                     try {
379                         fw.close();
380                     } catch (IOException e) {
381                     }
382                 }
383             }
384         }
385     }
386     if (responseXML != null) {
387         String postfix = ".xml";
388         String dataFormat = reqHeaders
389             .getProperty("X-EBay-SOA-RESPONSE-DATA-FORMAT");
390         if (dataFormat == null)
391             dataFormat = "XML";
392         if (dataFormat.equalsIgnoreCase("JSON")) {
393             postfix = ".json";
394             try {
395                 responseXML = new JSONObject(responseXML).toString(2);
396             } catch (JSONException e) {
397             }
398         }
399         try {
400             FileHelper.writeFile(outputDir + "/xmls/" + outputfilename
401                                 + postfix, responseXML);
402         } catch (IOException e) {
403             e.printStackTrace();
404         }
405         sbMessage.append("<a href=\"" + subFolder + "/xmls/" +
406                         + outputfilename + postfix + "\" target=_output></a>");
407     }
408     if (reqHeaders != null) {
409         String clogId = null;
410         if (reqHeaders.getProperty("X-Ebay-API-CAL-TRANSACTION-ID") != null)
411             clogId = reqHeaders
412                 .getProperty("X-Ebay-API-CAL-TRANSACTION-ID");
413         else {
414             clogId = reqHeaders.getProperty("X-Ebay-SOA-RLOGID");
415         }
416         if (clogId != null) {
417             if (ContextManager.getThreadContext().isCALlogEnabled())
418                 String calURL = CALHelper.getCalURL(clogId);
419                 sbMessage.append("<a href=\"" + calURL
420                                 + "\" target=cal:<logId/></a>");
421         }
422     }
423 }
424
425 Differences diff = compareGoldenFile(goldenFile, responseXML,
426                                     outputfilename, sbMessage, valCompareConfig);
427 Logging.logPICTCALC("", getAPIRequestName(reqHeaders, requestXML));
428 diff.setSbMessage(sbMessage.toString(), false);
429 if (diff != null) {
430     Assertion.assertTrue(
431         (diff.size() == 0),
432         "There are "
433         + diff.size()
434         + " differences between output and golden file, please check.");
435 }
436 return sbMessage.toString();
437 }
438
439 private static String getAPIRequestName(Properties reqHeaders,
440                                         String valRequest) {
441     String xmlRequestName = "";
442     try {
443         if (reqHeaders != null && !reqHeaders.isEmpty()) {
444             if (!reqHeaders.getProperty("X-Ebay-SOA-OPERATION-NAME") == null)
445                 xmlRequestName = reqHeaders
446                     .getProperty("X-Ebay-SOA-OPERATION-NAME")
447                     .replace("Request", "");
448             else if (!reqHeaders.getProperty("X-Ebay-API-CALL-NAME") == null)
449                 xmlRequestName = reqHeaders
450                     .getProperty("X-Ebay-API-CALL-NAME") + "Request";
451             else
452                 xmlRequestName = getXMLTitle(xmlRequest);
453         }
454     } catch (Exception ex) {
455         ex.printStackTrace();
456     }
457     return xmlRequestName;
458 }
459
460 protected static String getXMLTitle(String xmlString) {
461     String xmTitle = "";
462     String startPattern = "<$Body>";
463     String endPattern = "</xmls>";
464     if (xmString.indexOf(startPattern) > 0) {
465         int startIndex = xmString.indexOf(startPattern) + 13;
466         int endIndex = xmString.indexOf(endPattern, startIndex);
467         if (endIndex > -1) {
468             xmTitle = xmString.substring(startIndex, endIndex);
469             xmTitle = xmTitle.replace("<", "< ");
470             xmTitle = xmTitle.replace(">", "> ");
471             xmTitle = xmTitle.trim();
472         }
473     } else {
474         Pattern requestType = Pattern.compile("<(\\w+)\\>requestTypes");
475         Matcher matcher = requestType.matcher(xmString);
476         if (matcher.find()) {
477             xmTitle = matcher.group().replace("<", "< ")
478                         .replace(">", "> ");
479         }
480     }
481     System.out.println("XML title : " + xmTitle);
482     return xmTitle;
483 }
484
485 public static Differences compareGoldenFile(String goldenFile,
486

```

This screenshot shows a Java code editor within an IDE. The code is part of a class named `DefaultXmlSuite`. It contains several methods, including `return xmllitle;`, `public static Differences compareGoldenfile(String goldenfile, String responseXML, String outputDir, String outputfilename, StringDiffConfig sbMessage, Config xmlCompareConfig);`, and `Differences diff = null;`. The code involves reading files from a specified directory, writing to temporary files, and comparing them using a `XMLDog` object. It also handles exceptions and prints stack traces.

```
499     return xmllitle;
499
499 }
499
499 public static Differences compareGoldenfile(String goldenfile,
499     String responseXML, String outputDir, String outputfilename,
499     StringDiffConfig sbMessage, Config xmlCompareConfig) {
499     if (goldenfile != null && !goldenfile.isEmpty()) {
499
499         // Save API responses to gold fold
499         goldenfile = goldenfile.replaceFirst("\\\\", "/");
499
499         if (!new File(goldenfile).getParentFile().exists()) {
499             new File(goldenfile).getParentFile().mkdirs();
499         }
499
499         if (!new File(goldenfile).exists()) {
499             try {
499                 FileHelper.writeToFile(goldenfile, responseXML);
499             } catch (IOException e) {
499                 e.printStackTrace();
499             }
499         }
499     }
499
500     String xmloffputfile = outputDir + "/xmls/" + outputfilename
500     + ".xml";
500     xmloffputfile = xmloffputfile.replaceAll("\\\\", "/");
500
500     sbMessage.append("<a href=\"file://" + goldenfile
500         + "\">" + goldenfile
500         + "</a><script>copyFile("
500         + xmloffputfile + "," + goldenfile
500         + ")</script><span style='float:right; margin-right:10px;'>Save as golden</span>");
500
500     Differences diff = null;
500     XMLDog dog = null;
500
500     try {
500         if (xmlCompareConfig != null) {
500
500             // xmlCompareConfig.setCustomDifference(false);
500             // xmlCompareConfig.setApplyListToSibling(true);
500             // xmlCompareConfig.setIgnoringOrder(false);
500             dog = new XMLDog(xmlCompareConfig);
500         } else {
500             dog = new XMLDog();
500         }
500         diff = dog.compare(goldenfile, xmloffputfile);
500
500     } catch (Exception e) {
500         e.printStackTrace();
500     }
500
500     sbMessage.append("<a href=\"#" + "javadoc:toggle("
500         + outputfilename + "") + "\" class='difflink'>xml diff ("
500         + (diff != null ? diff.getDiffCount() : "UNKNOWN")
500         + ")</a><br>");
500     sbMessage.append("<div id=\"" + outputfilename
500         + "\" class='xmlDiff'>" + (diff == null ? "" : diff.getHTML()) + "</div>");
500
500     return diff;
500
500 } else {
500     return null;
500 }
500
500 }
```

This screenshot shows a Java code editor within an IDE. The code is part of a class named `DefaultTestNGCo`. It contains several methods, including `public void assertEquals(boolean actual, boolean expected, String message)`, `public void assertEquals(byte[] actual, byte[] expected, String message)`, and `public void assertEquals(char actual, char expected, String message)`. The code uses the `Assert` class for assertions and handles exceptions by adding them to a map of verification failures.

```
18 public class SoftAssertion {
19
20     private Map<ITestResult, List<Throwable>> verificationFailuresMap = new HashMap<ITestResult, List<Throwable>>();
21
22     public SoftAssertion() {
23
24     }
25
26     public SoftAssertion(Map<ITestResult, List<Throwable>> verificationFailuresMap) {
27         this.verificationFailuresMap = verificationFailuresMap;
28     }
29
30     public Map<ITestResult, List<Throwable>> getVerificationFailuresMap() {
31         return verificationFailuresMap;
32     }
33
34     protected void addVerificationFailure(Throwable e) {
35         List<Throwable> verificationFailures = getVerificationFailures();
36         verificationFailuresMap.put(Reporter.getCurrentTestResult(), verificationFailures);
37         verificationFailures.add(e);
38         Logging.logNull("ASSERTION FAILED: " + e.getMessage(), true, true);
39     }
40
41     ////////////////// Assertion Methods //////////////////
42
43     public void assertEquals(boolean actual, boolean expected, String message) {
44         try {
45             Assert.assertEquals(actual, expected, message);
46         } catch (Throwable e) {
47             _addVerificationFailure(e);
48         }
49     }
50
51     public void assertEquals(byte[] actual, byte[] expected, String message) {
52         try {
53             Assert.assertEquals(actual, expected, message);
54         } catch (Throwable e) {
55             _addVerificationFailure(e);
56         }
57     }
58
59     public void assertEquals(char[] actual, char[] expected, String message) {
60         try {
61             Assert.assertEquals(actual, expected, message);
62         } catch (Throwable e) {
63             _addVerificationFailure(e);
64         }
65     }
66
67     public void assertEquals(char actual, char expected, String message) {
68         try {
69             Assert.assertEquals(actual, expected, message);
70         } catch (Throwable e) {
71             _addVerificationFailure(e);
72         }
73     }
74
75     @SuppressWarnings("rawtypes")
76     public void assertEquals(Collection actual, Collection expected, String message) {
77         try {
78             Assert.assertEquals(actual, expected, message);
79         } catch (Throwable e) {
80             _addVerificationFailure(e);
81         }
82     }
83 }
```

Java EE Debug

```

1 DefaultTestNGCo  DefaultXmlSuite  IContextAttribut  IConverter.clas  Keyword.class  KeywordProvider  SoftAssertion.c  TearDownService  TestParameter.c  13
2 maucOMPONENT-eBayIndia  /Users/gaurshukla/m2/raptor2/com/ebay/maui/mauiplatform/1.2.1/mauiplatform-1.2.1.jar  com.ebay.maui.controller  SoftAssertion

83     public void assertEquals(double actual, double expected, String message) {
84         try {
85             Assert.assertEquals(actual, expected, message);
86         } catch (Throwable e) {
87             _addVerificationFailure(e);
88         }
89     }
90
91     public void assertEquals(Float actual, float expected, String message) {
92         try {
93             Assert.assertEquals(actual, expected, message);
94         } catch (Throwable e) {
95             _addVerificationFailure(e);
96         }
97     }
98
99     public void assertEquals(int actual, int expected, String message) {
100        try {
101            Assert.assertEquals(actual, expected, message);
102        } catch (Throwable e) {
103            _addVerificationFailure(e);
104        }
105    }
106
107    public void assertEquals(long actual, long expected, String message) {
108        try {
109            Assert.assertEquals(actual, expected, message);
110        } catch (Throwable e) {
111            _addVerificationFailure(e);
112        }
113    }
114
115    public void assertEquals(Object actual, Object expected, String message) {
116        try {
117            Assert.assertEquals(actual, expected, message);
118        } catch (Throwable e) {
119            _addVerificationFailure(e);
120        }
121    }
122
123    public void assertEquals(Object[] actual, Object[] expected, String message) {
124        try {
125            Assert.assertEquals(actual, expected, message);
126        } catch (Throwable e) {
127            _addVerificationFailure(e);
128        }
129    }
130
131    public void assertEquals(short actual, short expected, String message) {
132        try {
133            Assert.assertEquals(actual, expected, message);
134        } catch (Throwable e) {
135            _addVerificationFailure(e);
136        }
137    }
138
139    public void assertEquals(String actual, String expected, String message) {
140        try {
141            Assert.assertEquals(actual, expected, message);
142        } catch (Throwable e) {
143            _addVerificationFailure(e);
144        }
145    }
146
147

```

Read-Only Smart Insert 1 : 1

Java EE Debug

```

1 DefaultTestNGCo  DefaultXmlSuite  IContextAttribut  IConverter.clas  Keyword.class  KeywordProvider  SoftAssertion.c  TearDownService  TestParameter.c  13
2 maucOMPONENT-eBayIndia  /Users/gaurshukla/m2/raptor2/com/ebay/maui/mauiplatform/1.2.1/mauiplatform-1.2.1.jar  com.ebay.maui.controller  SoftAssertion

147     public boolean assertEquals2(String actual, String expected, String message) {
148         try {
149             Assert.assertEquals(actual, expected, message);
150         } catch (Throwable e) {
151             _addVerificationFailure(e);
152             return false;
153         }
154         return true;
155     }
156
157     public void assertFalse(boolean condition, String message) {
158         try {
159             Assert.assertFalse(condition, message);
160         } catch (Throwable e) {
161             _addVerificationFailure(e);
162         }
163     }
164
165     public void assertNotNull(Object object, String message) {
166         try {
167             Assert.assertNotNull(object, message);
168         } catch (Throwable e) {
169             _addVerificationFailure(e);
170         }
171     }
172
173     public void assertNotSame(Object actual, Object expected, String message) {
174         try {
175             Assert.assertNotSame(actual, expected, message);
176         } catch (Throwable e) {
177             _addVerificationFailure(e);
178         }
179     }
180
181     public void assertNull(Object object, String message) {
182         try {
183             Assert.assertNull(object, message);
184         } catch (Throwable e) {
185             _addVerificationFailure(e);
186         }
187     }
188
189     public void assertSame(Object actual, Object expected, String message) {
190         try {
191             Assert.assertSame(actual, expected, message);
192         } catch (Throwable e) {
193             _addVerificationFailure(e);
194         }
195     }
196
197     public void assertTrue(boolean condition, String message) {
198         try {
199             Assert.assertTrue(condition, message);
200         } catch (Throwable e) {
201             _addVerificationFailure(e);
202         }
203     }
204
205     public void fail(String message) {
206         Assert.fail(message);
207     }
208
209     public void logAll(String message) {
210         if (_verificationFailureMon != null && _verificationFailuresMon.isFaulty())
211             _verificationFailureMon.logAll(message);
212     }

```

Read-Only Smart Insert 1 : 1

```

159     public void assertFalse(boolean condition, String message) {
160         try {
161             Assert.assertFalse(condition, message);
162         } catch (Throwable e) {
163             _addVerificationFailure(e);
164         }
165     }
166     public void assertNull(Object object, String message) {
167         try {
168             Assert.assertNull(object, message);
169         } catch (Throwable e) {
170             _addVerificationFailure(e);
171         }
172     }
173     public void assertNotSame(Object actual, Object expected, String message) {
174         try {
175             Assert.assertSame(actual, expected, message);
176         } catch (Throwable e) {
177             _addVerificationFailure(e);
178         }
179     }
180     public void assertNotNull(Object object, String message) {
181         try {
182             Assert.assertNotNull(object, message);
183         } catch (Throwable e) {
184             _addVerificationFailure(e);
185         }
186     }
187     public void assertEquals(Object actual, Object expected, String message) {
188         try {
189             Assert.assertEquals(actual, expected, message);
190         } catch (Throwable e) {
191             _addVerificationFailure(e);
192         }
193     }
194     public void assertTrue(boolean condition, String message) {
195         try {
196             Assert.assertTrue(condition, message);
197         } catch (Throwable e) {
198             _addVerificationFailure(e);
199         }
200     }
201     public void fail(String message) {
202         Assert.fail(message);
203     }
204     public void logAll(String message) {
205         if (_verificationFailuresMap != null && !_verificationFailuresMap.isEmpty()) {
206             Assert.fail(message);
207         }
208     }
209     public List<Throwable> getVerificationFailures() {
210         List<Throwable> verificationFailures = _verificationFailuresMap.get(_reporter.getCurrentTestResult());
211         return verificationFailures == null ? new ArrayList<Throwable>() : verificationFailures;
212     }
213     public void tearDown() {
214         _reporter.tearDown();
215     }
216 }

```

```

1 package com.ebay.mau.controller;
2
3 /**
4 * This interface provides a way for test author to cleanup resources
5 * after a test method is completed executing.
6 */
7 public interface TearDownService {
8
9     public void tearDown();
10 }

```

```

1 package com.ebay.mauli.controller;
2
3 import java.lang.annotation.ElementType;
4 import java.lang.annotation.Retention;
5 import java.lang.annotation.RetentionPolicy;
6 import java.lang.annotation.Target;
7
8 @Retention(RetentionPolicy.RUNTIME)
9 @Target(ElementType.FIELD)
10 public @interface TestParameter {
11
12     /**
13      * The XML parameter name of the testNG.xml file or system property
14      */
15     String name();
16 }
17
18

```

```

29 public abstract class TestPlan {
30     private static final Logger logger = Logging.getLogger(TestPlan.class);
31     private Date start;
32
33     static {
34     }
35
36     /**
37      * Implement an anonymous TearDownService and add it to the framework. The
38      * framework will execute it once the test is complete whether it's passed
39      * or failed does not matter.
40      *
41      * One example can be recycling of user. <code>
42      * <pre>
43      *     addtearDownService(new TearDownService() {
44      *         public void tearDown() {
45      *             UserHelper.recycleUser();
46      *         }
47      *     });
48      *     UserHelper.createUser();
49      * </pre>
50      * </code>
51      *
52      * @param service
53      */
54     public static void addtearDownService(TearDownService service) {
55         ContextManager.getThreadContext().addtearDownService(service);
56     }
57
58     @AfterMethod(alwaysRun = true)
59     public void afterTestMethod(Method method) {
60         // Clean up services
61         List<TearDownService> serviceList = ContextManager.getThreadContext()
62             .gettearDownServices();
63
64         if (serviceList != null && !serviceList.isEmpty()) {
65             for (TearDownService service : serviceList) {
66                 service.tearDown();
67             }
68         }
69         Thread.currentThread().setMethodtearDownServiceList(null);
70         WebNDriver.cleanup();
71         APIDriver.cleanup();
72         DBDriver.cleanup();
73         SSIDriver.cleanup();
74         logger.info(Thread.currentThread() + " Finish method "
75             + method.getName());
76     }
77
78     @AfterSuite(alwaysRun = true)
79     public void afterTestSuite() {
80         logger.info("Test Suite Execution Time: "
81             + (System.currentTimeMillis() - start.getTime()) / 1000 / 60
82             + " minutes.");
83     }
84
85     /**
86      * Configure Test Params setting
87      *
88      * @param xmTest
89      */
90     @BeforeTest(alwaysRun = true)
91     public void beforeTest(ITestContext testContex, XmlTest xmTest) {
92         ContextManager.initTestLevelContext(testContex, xmTest);
93     }
94

```

```

TestPlan.class <--> EasyFilter.class <--> SpreadSheetUtil <--> CheckClaimTestP <--> AbstractPageLis <--> AbstractPageLs <--> AnnotationTrans <--> Assertion.class <--> Context.class >>11 Java EE Debug
mauicomponent-ebayindia <--> /Users/gursukha/m2/raptor2/com/ebay/maui/maulplatform/1.2.1/maulplatform-1.2.1.jar <--> com.ebay.maui.controller <--> TestPlan <--> logger : Logger

94    @BeforeSuite(suiteName = "ebayindia")
95    public void beforeSuite(@TestContext xmlTest, Method method,
96                           ITestContext testContext, XMLTest xmlTest) {
97        logger.info("Thread.currentThread() = " + Start method)
98        logger.info("method.getName() = " + method.getName());
99        ContextManager.initThreadContext(testContext, xmlTest);
100
101        if (method != null) {
102            ContextManager.getThreadContext().setAttribute(
103                context.TEST_METHOD_SIGNATURE,
104                constructMethodSignature(method, parameters));
105        }
106    }
107
108
109    @BeforeSuite(suiteName = true)
110    public void beforeSuite(ITestContext testContext) {
111        start = new Date();
112        System.setProperty("AUTOMATION_FRAMEWORK", "MAUI");
113        ContextManager.initGlobalContext(testContext);
114
115        // to support to call function in @beforeSuite
116        ContextManager.initThreadContext(testContext, null);
117
118        readTestParameterAnnotations();
119    }
120
121    private String constructMethodSignature(Method method, Object[] parameters) {
122        return method.getDeclaringClass().getCanonicalName() + "."
123            + method.getName() + "(" + constructParameterString(parameters)
124            + ")";
125    }
126
127
128    /**
129     * Remove name space from parameters
130     * @param parameters
131     * @return
132     */
133    private String constructParameterString(Object[] parameters) {
134        StringBuffer sbParam = new StringBuffer();
135
136        if (parameters != null) {
137            for (int i = 0; i < parameters.length; i++) {
138                if (parameters[i] == null) {
139                    sbParam.append("null,");
140                } else if (parameters[i] instanceof java.lang.String) {
141                    sbParam.append("\"").append(parameters[i]).append("\", ");
142                } else {
143                    sbParam.append(parameters[i]).append(", ");
144                }
145            }
146        }
147
148        if (sbParam.length() > 0)
149            sbParam.delete(sbParam.length() - 2, sbParam.length() - 1);
150
151        return sbParam.toString();
152    }
153
154
155    /**
156     * Get the value of property with TestParameter annotation from system
157     * property, system env, or Testing parameter
158     */
159
160    protected void readTestParameterAnnotations() {
161        try {
162            Field[] fields = this.getClass().getDeclaredFields();
163            AccessibleObject.setAccessible(fields, true);
164
165            for (Field field : fields) {
166                if (field.getAnnotation(TestParameter.class) != null) {
167                    String name = field.getAnnotation(TestParameter.class)
168                        .name();
169
170                    if (System.getProperty(name) != null) {
171                        field.set(this, System.getProperty(name));
172                    } else if (System.getenv(name) != null) {
173                        field.set(this, System.getenv(name));
174                    } else if (ContextManager.getThreadContext()
175                               .getTestNGContext().getCurrentXmlTest()
176                               .getParameter(name) != null) {
177                        field.set(this, ContextManager.getThreadContext()
178                               .getTestNGContext().getCurrentXmlTest()
179                               .getParameter(name));
180                    }
181                }
182            }
183        } catch (Exception ex) {
184        }
185    }
186

```

The screenshot shows the Java code for `TestRetryAnalyzer` in an IDE. The code implements the `IRetryAnalyzer` interface and handles test retries based on configuration properties and a maximum count.

```
1 package com.ebay.maui.controller;
2
3 import org.testng.IRetryAnalyzer;
4 import org.testng.ITestResult;
5
6 /**
7 * Implement to be able to have a chance to retry a failed test
8 */
9 * @author bshen
10 */
11 public class TestRetryAnalyzer implements IRetryAnalyzer {
12     private static final String TEST_RETRY_COUNT = "testRetryCount";
13     private static final String TEST_RETRY_INTERVAL = "testRetryInterval";
14     private int count = 1;
15     private int maxCount = 1;
16
17     public TestRetryAnalyzer() {
18         String retryCount = System.getProperty(TEST_RETRY_COUNT);
19         if (retryCount != null) {
20             maxCount = Integer.parseInt(retryCount);
21         }
22     }
23
24     public void setMaxCount(int count) {
25         this.maxCount = count;
26     }
27
28     public int getCount() {
29         return this.count;
30     }
31
32     public int getMaxCount() {
33         return this.maxCount;
34     }
35
36     public synchronized boolean retry(ITestResult result) {
37         String testClassName = String.format("%s.%s", result.getMethod()
38             .getRealClass().toString(), result.getMethod().getMethodName());
39
40         if (count <= maxCount) {
41             result.setAttribute("RETRY", new Integer(count));
42
43             Logging.log("RETRYING " + testClassName + " FAILED, "
44                     + "Retrying " + count + " time", true);
45             count += 1;
46
47             String retryInterval = System.getProperty(TEST_RETRY_INTERVAL);
48             if (retryInterval != null) {
49                 try {
50                     int intervalSeconds = Integer.parseInt(retryInterval);
51                     Logging.log("Retry after " + intervalSeconds + " seconds",
52                             true);
53                     Thread.sleep(intervalSeconds * 1000);
54                 } catch (Exception ex) {
55                 }
56             }
57         }
58         return true;
59     }
60     return false;
61 }
62 }
```

The screenshot shows the Java code for `TestRetryListener` in an IDE. It implements the `IAnnotationTransformer` interface to handle test retries at runtime.

```
1 package com.ebay.maui.controller;
2
3 import java.lang.reflect.Constructor;
4 import java.lang.reflect.Method;
5
6 import org.testng.IAnnotationTransformer;
7 import org.testng.IRetryAnalyzer;
8 import org.testng.annotations.ITestAnnotation;
9
10 /**
11  * TestRetryListener is a TestNG Listener to enable Retry at runtime
12 */
13 * @author bshen
14 */
15 public class TestRetryListener implements IAnnotationTransformer {
16     @Override
17     public void transform(ITestAnnotation annotation, Class testClass, Constructor testConstructor, Method testMethod) {
18         IRetryAnalyzer retryAnalyzer = annotation.getRetryAnalyzer();
19         if (retryAnalyzer == null) {
20             annotation.setRetryAnalyzer(TestRetryAnalyzer.class);
21         }
22     }
23 }
```

The Project Explorer on the left shows the project structure, including the `mauiplatform-1.2.1.jar` file and various source files like `AbstractPageListener.class`, `AnnotationTransformer.class`, and `TestRetryAnalyzer.class`.