

Screenshot of Eclipse IDE showing the Java EE perspective with the API Driver class highlighted.

**Project Explorer:**

- META-INF
- tmp
- maulplatform-1.2.1.jar - /Users/gaurshukla/m2/repo
- com.ebay.maul.controller
  - APIDriver.class
  - APIRequest.class
  - APIResponse.class
  - EasySSLProtocolSocketFactory.class
  - EasyX509TrustManager.class
  - HttpMethod.class
  - RequestFilter.class
  - ResponseFactory.class
  - RestClient.class
  - RestClientReportLogger.class
  - RESTResponse.class
  - com.ebay.maul.driver.db
  - com.ebay.maul.driver.email
  - com.ebay.maul.driver.ssh
  - com.ebay.maul.driver.web
    - BaseWebUtil.class
    - BrowserType.class
    - CustomEventFiringWebDriver.class
    - IScreenShotListener.class
    - ScreenShot.class
    - ScreenShotRemoteWebDriver.class
    - ScreenshotUtil.class
    - WebDriverConfig.class
    - WebDriverExceptionListener.class
    - WebDriverMode.class
    - WebUIXDriver.class
    - com.ebay.maul.driver.web.element
      - BaseHtmlPage.class
      - BasePageObject.class
      - Button.class
      - CheckBox.class
      - Form.class
      - GenericWebResponse.class
      - Image.class
      - IPage.class
      - Label.class
      - Link.class
      - LocatorSwitch.class
      - LocatorType.class

**Java Editor (Test.class):**

```

1 package com.ebay.maul.driver.apl;
2
3 import java.io.BufferedReader;
4 import java.io.IOException;
5 import java.util.Properties;
6 import java.util.zip.GZIPOutputStream;
7
8 import org.apache.commons.httpclient.HttpClient;
9 import org.apache.commons.httpclient.HttpConnection;
10 import org.apache.commons.httpclient.HttpException;
11 import org.apache.commons.httpclient.HttpMethod;
12 import org.apache.commons.httpclient.methods.PostMethod;
13 import org.apache.commons.httpclient.methods.StringRequestEntity;
14 import org.apache.commons.httpclient.protocol.Protocol;
15 import org.apache.commons.log.Log;
16 import org.apache.commons.log.LogFactory;
17
18 import com.ebay.maul.controller.Context;
19 import com.ebay.maul.controller.ContextManager;
20 import com.ebay.maul.controller.Logging;
21 import com.ebay.maul.exception.APITException;
22 import com.ebay.maul.helper.EscapeHelper;
23 import com.ebay.qa.xmldog.Config;
24
25 /**
26  * APIDriver provides ways for API/SoA testing.
27 */
28
29 public class APIDriver {
30     protected static final Log logger = Logging.getLogger(APIDriver.class);
31
32     private static ThreadLocal<APIDriver> threadLocalSession = new ThreadLocal<APIDriver>();
33
34     public static void cleanup() {
35         APIDriver driver = threadLocalSession.get();
36         if (driver != null) {
37             if (ContextManager.getThreadContext() != null)
38                 //logger.info(ContextManager.getThreadContext().getTestMethodSignature() + " - Cleaning up API Sessions");
39             driver.close();
40         }
41     }
42 }
43
44     private APIRequest request = null;
45     private APIResponse response = null;
46
47     public APIDriver() {
48         threadLocalSession.set(this);
49     }
50
51     public APIDriver(APIRequest request) {
52         this.request = request;
53         threadLocalSession.set(this);
54     }
55
56     public void close() {
57         reconnect = null;
58     }
59
60     public void setRequest(APIRequest request) {
61         this.request = request;
62     }
63
64     public APIResponse executeAPIRequest() throws APITException {
65         return response;
66     }
67
68     public void getRequest() {
69         response = null;
70     }
71
72     public void setResponse(APIResponse response) {
73         this.response = response;
74     }
75
76     public void generateLogMessage(String message) {
77         logger.info(message);
78     }
79
80     public void cleanUp() {
81         threadLocalSession.set(null);
82     }
83
84     public void closeUp() {
85         cleanup();
86     }
87
88     public void disconnect() {
89         reconnect = null;
90     }
91
92     public void reconnect() {
93         reconnect = true;
94     }
95
96     public void setThreadLocalSession(ThreadLocal<APIDriver> threadLocalSession) {
97         this.threadLocalSession = threadLocalSession;
98     }
99
100    public ThreadLocal<APIDriver> getThreadLocalSession() {
101        return threadLocalSession;
102    }
103}

```

**Call Hierarchy:**

Members calling constructors of 'WebPageSection' - in workspace

Test.class

```

58     public void close() {
59         request = null;
60         response = null;
61     }
62
63     @SupressesWarnings("deprecation")
64     public APIResponse executeAPIRequest() throws APIException {
65         Context context = ContextManager.getThreadContext();
66
67         if (request == null || !request.isValid()) {
68             throw new APIException("API request or endpoint can't be null");
69         }
70
71         String serverEndPoint = request.getServerEndPoint();
72         if (serverEndPoint.toLowerCase().startsWith("http://")) {
73             //String server = serverEndPoint.substring(serverEndPoint.indexOf("//") + 3, serverEndPoint.indexOf(".com") + 4);
74             Protocol.registerProtocol("https", new Protocol("https", new EasySSLProtocolSocketFactory(), 443));
75         }
76
77         HttpClient httpClient = new HttpClient();
78         httpClient.setHttpConnectionManager(context.getParams());
79         if (serverEndPoint.toLowerCase().contains("http://")) {
80             httpClient.getHostConfiguration().setProxyText(context.getApiProxyHost(), Integer.parseInt(context.getApiProxyPort()));
81         }
82
83         //Comment user agent handle os it does not work
84         //String pc = ContextManager.getThreadContext().getPCSettings();
85         //if (pc != null)
86         //    httpClient.getParams().setParameter(CoreProtocolNames.USER_AGENT, pc);
87
88         if (serverEndPoint.indexOf("?) < 0) {
89             serverEndPoint = serverEndPoint + "?";
90         }
91         if (serverEndPoint.endsWith("?")) {
92             serverEndPoint = serverEndPoint + "&";
93         }
94         if (request.getRequestParameters() != null) {
95             for (Entry<Object, Object> parameter : request.getRequestParameters().entrySet()) {
96                 serverEndPoint = serverEndPoint + parameter.getKey() + "=" + EscapeHelper.escapeHTML(parameter.getValue().toString());
97             }
98         }
99
100        PostMethod method = new PostMethod(serverEndPoint);
101        if (request.getRequestHeaders() != null) {
102            for (Entry<Object, Object> header : request.getRequestHeaders().entrySet()) {
103                method.setRequestHeader(header.getKey(), (String) header.getValue());
104            }
105        }
106
107        if (request.getRequestParameters() != null) {
108            for (Entry<Object, Object> parameter : request.getRequestParameters().entrySet()) {
109                method.setParameter((String) parameter.getKey(), EscapeHelper.escapeHTML(parameter.getValue().toString()));
110            }
111        }
112        if (request.getRequestXML().indexOf("<soapenv:Envelope") != -1 && (request.getRequestHeaders() == null || !request.getRequestHeaders().containsKey("SOAPAction"))) {
113            method.setRequestHeader("SOAPAction", "");
114        }
115
116        BufferedReader br = null;
117
118        int responseCode = 0;
119        Properties responseHeaders = new Properties();
120        StringBuffer xmlOutputSB = new StringBuffer();

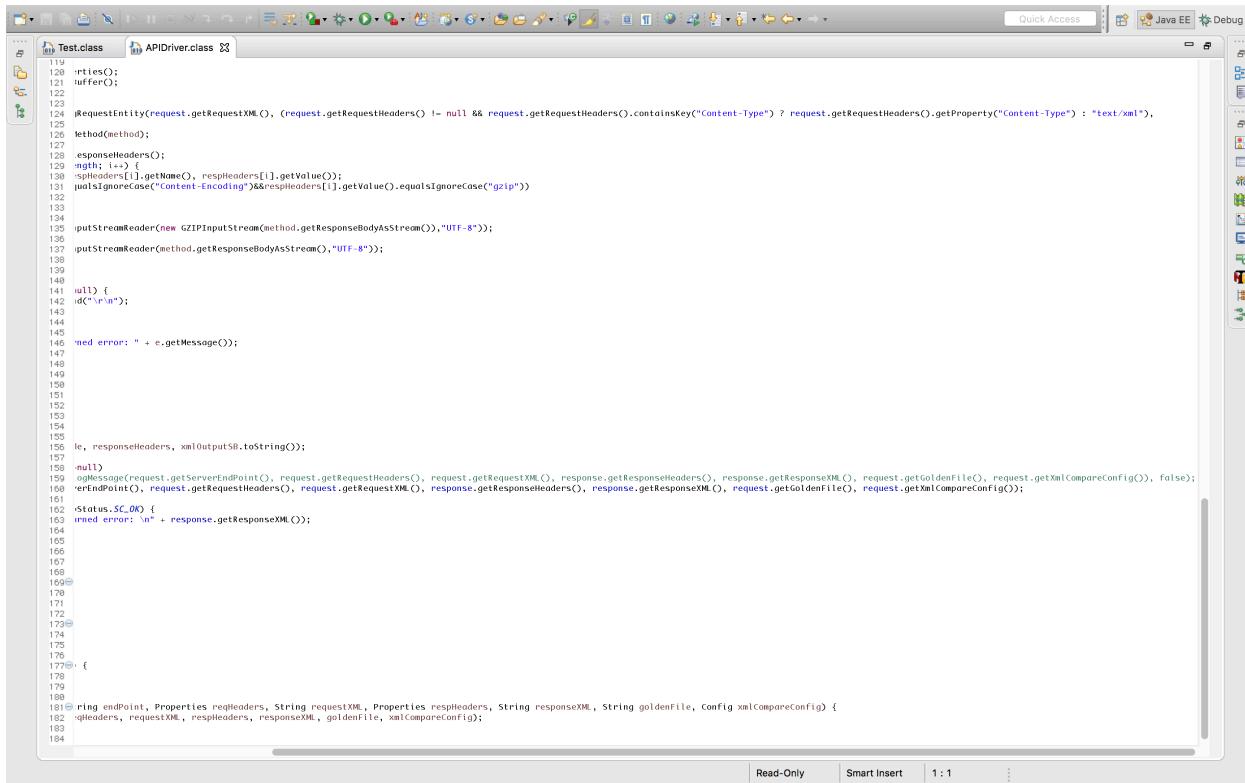
```

APIDriver.class

```

120        Properties responseHeaders = new Properties();
121        StringBuffer xmlOutputSB = new StringBuffer();
122
123        try {
124            method.setRequestEntity(new StringRequestEntity(request.getRequestXML(), (request.getRequestHeaders() != null & request.getRequestHeaders().containsKey("Content-Type")) ? request.getRequestHeaders().getPr
125                .get("Content-Type") : "UTF-8"));
126            responseCode = httpClient.executeMethod(method);
127            boolean isGZIP = false;
128            Header[] resHeaders = method.getResponseHeaders();
129            for (Header header : resHeaders) {
130                if (header.getName().equalsIgnoreCase("Content-Encoding")) {
131                    responseHeaders.setProperty(header.getName(), header.getValue());
132                    if (header.getName().equalsIgnoreCase("gzip") && header.getValue().equalsIgnoreCase("gzip"))
133                        isGZIP = true;
134                }
135            }
136            if (isGZIP) {
137                br = new BufferedReader(new InputStreamReader(new GZIPOutputStream(method.getResponseBodyAsStream()), "UTF-8"));
138            } else {
139                br = new BufferedReader(new InputStreamReader(method.getResponseBodyAsStream(), "UTF-8"));
140            }
141            String line = null;
142            while ((line = br.readLine()) != null) {
143                xmlOutputSB.append(line).append("\r\n");
144            }
145        } catch (Exception e) {
146            e.printStackTrace();
147            throw new APIException("Call returned error: " + e.getMessage());
148        } finally {
149            method.releaseConnection();
150            if (br != null)
151                try {
152                    br.close();
153                } catch (Exception fe) {
154                }
155        }
156
157        response = new APIResponse(responseCode, responseHeaders, xmlOutputSB.toString());
158
159        if (ContextManager.getThreadContext() != null)
160            Logging.logPSTCall("", generateLogMessage(request.getServerEndPoint(), request.getRequestHeaders(), request.getRequestXML(), response.getResponseHeaders(), response.getResponseXML(), request.getGoldenFi
161            Logging.logPSTStep(request.getServerEndPoint(), request.getRequestHeaders(), request.getRequestXML(), response.getResponseHeaders(), response.getResponseXML(), request.getGoldenFile(), request.getCompa
162
163        if (response.getResponseCode() != HttpStatus.SC_200)
164            throw new APIException("Call returned error: " + response.getResponseXML());
165
166
167        return response;
168    }
169    public APIRequest getRequest() {
170        return request;
171    }
172
173    public APIResponse getResponse() {
174        return response;
175    }
176
177    public void setRequest(APIRequest request) {
178        this.request = request;
179    }
180
181    public static String generateLogMessage(String endPoint, Properties reqHeaders, String requestXML, Properties respHeaders, String responseXML, String goldenFile, Config xmlCompareConfig) {
182        return Logging.logPSTStep(endPoint, reqHeaders, requestXML, respHeaders, responseXML, goldenFile, xmlCompareConfig);
183    }
184}
185

```



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

- Title Bar:** Quick Access, Java EE, Debug.
- Toolbars:** Standard Java toolbars.
- Left Sidebar:** Shows the project structure with files like `Test.class` and `APIDriver.class`.
- Code Editor:** Displays the following Java code:

```
119
120     writesO;
121     bufferO;
122
123     @RequestEntity(request.getRequestXML(), (request.getRequestHeaders() != null & request.getRequestHeaders().containsKey("Content-Type")) ? request.getRequestHeaders().getProperty("Content-Type") : "text/xml"),
124     method(method);
125
126     responseHeadersO;
127     #gith: i-->
128     responseHeaders[i].getNomeO, respHeaders[i].getValueO);
129     putsIgnoreCase("Content-Encoding")&respHeaders[i].getValueO.equalsIgnoreCase("gzip"))
130
131     132
133
134     inputStreamReader(new GZIPInputStream(method.getResponseBodyAsStream()), "UTF-8");
135     inputStreamReader(method.getResponseBodyAsStream(), "UTF-8");
136
137
138
139
140     null) {
141     idC+"\r\n");
142
143
144
145
146    imed_error: " + e.getMessage());
147
148
149
150
151
152
153
154
155     ie, responseHeaders, xmlOutputSB.toString());
156
157
158     null)
159     ogMessage(request.getServerEndPoint(), request.getRequestHeaders(), request.getRequestXML(), response.getResponseHeaders(), response.getResponseXML(), request.getGoldenFile(), request.getXmlCompareConfig(), false);
160     erEndpoint(), request.getRequestHeaders(), request.getRequestXML(), response.getResponseHeaders(), response.getResponseXML(), request.getGoldenFile(), request.getXmlCompareConfig());
161
162     Status.SC_00 {
163     imed_error: "\n" + response.getResponseXML();
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181     ring endPoint, Properties reqHeaders, String requestXML, Properties respHeaders, String responseXML, String goldenFile, Config xmlCompareConfig) {
182     qHeaders, requestXML, respHeaders, responseXML, goldenFile, xmlCompareConfig);
183
184
```

The code is annotated with line numbers from 119 to 184. It includes imports and annotations for `@RequestEntity`, `@Method`, and `@ResponseHeaders`. The code handles XML input and output, including gzip decompression and XML comparison logic.

The screenshot shows the Java code for the `APIRequest` class in an IDE. The code is annotated with line numbers from 1 to 66. It includes imports for `com.ebay.maul.driver.api`, `java.util.Properties`, and `com.ebay.qa.xmldog.Config`. The class has private fields for `serverEndPoint`, `requestParameters`, `requestHeaders`, `requestObj`, and `responseObj`. It contains several constructor-like methods: `APIRequest(String serverEndPoint, Properties parameters, Properties headers, Object requestObj)`, `APIRequest(String serverEndPoint, Properties parameters, Properties headers, String requestXML)`, and `APIRequest(String serverEndPoint, Properties parameters, Properties headers, String requestXML, String goldenFile, Config xmlCompareConfig)`. Other methods include `getGoldenfile()`, `getRequestHeaders()`, `getRequestObj()`, `getRequestParameters()`, `getRequestXML()`, `getResponseObj()`, `getServerEndPoint()`, and `isValid()`.

```
1 package com.ebay.maul.driver.api;
2
3 import java.util.Properties;
4
5 import com.ebay.qa.xmldog.Config;
6
7 public class APIRequest {
8     private String serverEndPoint = "";
9     private Properties requestParameters = null;
10    private Properties requestHeaders = null;
11    private String requestXML = null;
12    private String goldenFile = null;
13    private Config xmlCompareConfig = null;
14    private Object requestObj = null;
15    private Object responseObj = null;
16
17    public APIRequest(String serverEndPoint, Properties parameters, Properties headers, Object requestObj) {
18        this.serverEndPoint = serverEndPoint;
19        this.requestParameters = parameters;
20        this.requestHeaders = headers;
21        this.requestObj = requestObj;
22    }
23
24    public APIRequest(String serverEndPoint, Properties parameters, Properties headers, String requestXML) {
25        this.serverEndPoint = serverEndPoint;
26        this.requestParameters = parameters;
27        this.requestHeaders = headers;
28        this.requestXML = requestXML.replace("\n", "").trim() + "\n";
29    }
30
31    public APIRequest(String serverEndPoint, Properties parameters, Properties headers, String requestXML, String goldenFile, Config xmlCompareConfig) {
32        this.serverEndPoint = serverEndPoint;
33        this.requestParameters = parameters;
34        this.requestHeaders = headers;
35        this.requestXML = requestXML.replace("\n", "").trim() + "\n";
36        this.goldenfile = goldenFile.replace("\n", "").trim() + "\n";
37        this.xmlCompareConfig = xmlCompareConfig;
38    }
39
40    public String getGoldenfile() {
41        return goldenfile;
42    }
43
44    public Properties getRequestHeaders() {
45        return requestHeaders;
46    }
47
48    public Object getRequestObj() {
49        return requestObj;
50    }
51
52    public Properties getRequestParameters() {
53        return requestParameters;
54    }
55
56    public String getRequestXML() {
57        return requestXML;
58    }
59
60    public Object getResponseObj() {
61        return responseObj;
62    }
63
64    public String getServerEndPoint() {
65        return serverEndPoint;
66    }
67
68    public Config getXmLCompareConfig() {
69        return xmlCompareConfig;
70    }
71
72    public boolean isValid() {
73        if (getServerEndPoint() == null || "".equals(getServerEndPoint().trim()) || getRequestXML() == null || "".equals(getRequestXML().trim())) {
74            return false;
75        }
76        return true;
77    }
78
79    public void setGoldenFile(String goldenFile) {
80        this.goldenfile = goldenfile;
81    }
82
83    public void setRequestHeaders(Properties requestHeaders) {
84        this.requestHeaders = requestHeaders;
85    }
86
87    public void setRequestObj(Object requestObj) {
88        this.requestObj = requestObj;
89    }
90
91    public void setRequestParameters(Properties requestParameters) {
92        this.requestParameters = requestParameters;
93    }
94
95    public void setRequestXML(String requestXML) {
96        this.requestXML = requestXML;
97    }
98
99    public void setResponseObj(Object responseObj) {
100        this.responseObj = responseObj;
101    }
102
103    public void setServerEndPoint(String serverEndPoint) {
104        this.serverEndPoint = serverEndPoint;
105    }
106
107    public void setXmLCompareConfig(Config xmLCompareConfig) {
108        this.xmlCompareConfig = xmLCompareConfig;
109    }
110
111
112}
113
```

The screenshot shows the Java code for the `Test` class in an IDE. The code is annotated with line numbers from 48 to 113. It includes methods for setting and getting various properties and objects, such as `getRequestObj()`, `getRequestParameters()`, `getRequestXML()`, `getResponseObj()`, `getServerEndPoint()`, `getXmLCompareConfig()`, `isValid()`, `setGoldenFile()`, `setRequestHeaders()`, `setRequestObj()`, `setRequestParameters()`, `setRequestXML()`, `setResponseObj()`, `setServerEndPoint()`, and `setXmLCompareConfig()`.

```
48    public Object getRequestObj() {
49        return requestObj;
50    }
51
52    public Properties getRequestParameters() {
53        return requestParameters;
54    }
55
56    public String getRequestXML() {
57        return requestXML;
58    }
59
60    public Object getResponseObj() {
61        return responseObj;
62    }
63
64    public String getServerEndPoint() {
65        return serverEndPoint;
66    }
67
68    public Config getXmLCompareConfig() {
69        return xmLCompareConfig;
70    }
71
72    public boolean isValid() {
73        if (getServerEndPoint() == null || "".equals(getServerEndPoint().trim()) || getRequestXML() == null || "".equals(getRequestXML().trim())) {
74            return false;
75        }
76        return true;
77    }
78
79    public void setGoldenFile(String goldenFile) {
80        this.goldenfile = goldenfile;
81    }
82
83    public void setRequestHeaders(Properties requestHeaders) {
84        this.requestHeaders = requestHeaders;
85    }
86
87    public void setRequestObj(Object requestObj) {
88        this.requestObj = requestObj;
89    }
90
91    public void setRequestParameters(Properties requestParameters) {
92        this.requestParameters = requestParameters;
93    }
94
95    public void setRequestXML(String requestXML) {
96        this.requestXML = requestXML;
97    }
98
99    public void setResponseObj(Object responseObj) {
100        this.responseObj = responseObj;
101    }
102
103    public void setServerEndPoint(String serverEndPoint) {
104        this.serverEndPoint = serverEndPoint;
105    }
106
107    public void setXmLCompareConfig(Config xmLCompareConfig) {
108        this.xmlCompareConfig = xmLCompareConfig;
109    }
110
111
112}
113
```

The screenshot shows the Eclipse IDE interface with the Java EE perspective selected. The central view displays the code for the `APIResponse` class. The code is as follows:

```

package com.ebay.mau.driver.api;
import java.util.Properties;
public class APIResponse {
    private int responseCode;
    private Properties responseHeaders = null;
    private String responseXML = null;
    public APIResponse() {
    }
    public APIResponse(int responseCode, Properties responseHeaders, String responseXML) {
        this.responseCode = responseCode;
        this.responseHeaders = responseHeaders;
        this.responseXML = responseXML;
    }
    public int getResponseCode() {
        return responseCode;
    }
    public Properties getResponseHeaders() {
        return responseHeaders;
    }
    public String getResponseXML() {
        return responseXML;
    }
    public void setResponseCode(int responseCode) {
        this.responseCode = responseCode;
    }
    public void setResponseHeaders(Properties responseHeaders) {
        this.responseHeaders = responseHeaders;
    }
    public void setResponseXML(String responseXML) {
        this.responseXML = responseXML;
    }
}

```

The right side of the interface shows the `APIResponse` class in the Call Hierarchy view, listing methods like `getProperties()`, `getHeaders()`, and `getXML()`. The bottom status bar indicates "Read-Only" and "Smart Insert".

The screenshot shows the Eclipse IDE interface with the Java EE perspective selected. The central view displays the code for the `EasySSLProtocolSocketFactory` class. The code is as follows:

```

package com.ebay.mau.driver.api;
import java.io.IOException;
import java.net.InetAddress;
import java.net.Socket;
import java.net.UnknownHostException;
import javax.net.ssl.SSLContext;
import javax.net.ssl.TrustManager;
import org.apache.commons.httpclient.ConnectTimeoutException;
import org.apache.commons.httpclient.HttpClientError;
import org.apache.commons.httpclient.params.HttpConnectionParams;
import org.apache.commons.httpclient.protocol.ControllerThreadSocketFactory;
import org.apache.commons.httpclient.protocol.SecureProtocolSocketFactory;
public class EasySSLProtocolSocketFactory implements SecureProtocolSocketFactory {
    private static SSLContext createEasySSLContext() {
        try {
            SSLContext context = SSLContext.getInstance("SSL");
            context.init(null, new TrustManager[] { new EasyX509TrustManager(null) }, null);
        } catch (Exception e) {
            throw new HttpClientError(e.toString());
        }
    }
    private SSLContext sslcontext = null;
    /**
     * Constructor for EasySSLProtocolSocketFactory.
     */
    public EasySSLProtocolSocketFactory() {
        super();
    }
    /**
     * @see SecureProtocolSocketFactory#createSocket(Socket,java.lang.String,int,boolean)
     */
    public Socket createSocket(Socket socket, String host, int port, boolean autoClose) throws IOException, UnknownHostException {
        return getSSLContext().getSocketFactory().createSocket(socket, host, port, autoClose);
    }
    /**
     * @see SecureProtocolSocketFactory#createSocket(java.lang.String,int)
     */
    public Socket createSocket(String host, int port) throws IOException, UnknownHostException {
        return getSSLContext().getSocketFactory().createSocket(host, port);
    }
    /**
     * @see SecureProtocolSocketFactory#createSocket(java.lang.String,int,java.net.InetAddress,int)
     */
    public Socket createSocket(String host, int port, InetAddress clientHost, int clientPort) throws IOException, UnknownHostException {
        return getSSLContext().getSocketFactory().createSocket(host, port, clientHost, clientPort);
    }
    /**
     * Attempts to get a new socket connection to the given host within the given time limit.
     */
    To circumvent the limitations of older JREs that do not support connect timeout a controller thread is executed. The controller thread attempts to create a new socket within the given limit of time. If socket timeout expires, the controller terminates and throws an @link ConnectTimeoutException
    */
}

```

The bottom status bar indicates "Read-Only" and "Smart Insert".

The screenshot shows the Java code for the `Test.class` file in an IDE. The code is annotated with various Javadoc-style comments and annotations. The annotations include `@param`, `@throws`, and `/*` blocks. The code is part of a class named `EasySSLProtocolSocketFactory`.

```
48  public Socket createSocket(String host, int port) throws IOException, UnknownHostException {
49      return getSSLContext().getSocketFactory().createSocket(host, port);
50  }
51  /**
52   * @see SecureProtocolSocketFactory#createSocket(java.lang.String,int,java.net.InetAddress,int)
53   */
54  public Socket createSocket(String host, int port, InetAddress clientHost, int clientPort) throws IOException, UnknownHostException {
55      return getSSLContext().getSocketFactory().createSocket(host, port, clientHost, clientPort);
56  }
57  /**
58   * Attempts to get a new socket connection to the given host within the given time limit.
59   * <p>
60   * If older JREs that do not support connect timeout a controller thread is executed. The controller thread attempts to create a new socket within the given limit of time. If socket
61   * timeout expires, the controller terminates and throws on @link ConnectTimeoutException
62   * </p>
63   * @param host
64   *     the host name/IP
65   * @param port
66   *     the port on the host
67   * @param clientHost
68   *     the local host name/IP to bind the socket to
69   * @param clientPort
70   *     the port on the local machine
71   * @param params
72   *     @param params
73   *         (Only HttpConnectionParams Http connection parameters)
74   * @throws IOException
75   *     if an I/O error occurs while creating the socket
76   * @throws UnknownHostException
77   *     if the IP address of the host cannot be determined
78   */
79  public Socket createSocket(final String host, final int port, final InetAddress localAddress, final int localPort, final HttpConnectionParams params) throws IOException, UnknownHostException, ConnectTimeoutException {
80      if (params == null) {
81          throw new IllegalArgumentException("Parameters may not be null");
82      }
83      int timeout = params.getConnectionTimeout();
84      if (timeout > 0) {
85          return createSocket(host, port, localAddress, localPort);
86      } else {
87          // To be eventually deprecated when migrated to Java 1.4 or above
88          return ControllerThreadSocketFactory.createSocket(this, host, port, localAddress, localPort, timeout);
89      }
90  }
91  /**
92   * Returns a new socket
93   * @throws IOException
94   */
95  public boolean equals(Object obj) {
96      return (obj != null) && obj.getClass().equals(EasySSLProtocolSocketFactory.class);
97  }
98  /**
99   * Gets the SSLContext
100  * @return the SSLContext
101  */
102  private SSLContext getSSLContext() {
103      if (this.sslContext == null) {
104          this.sslContext = createEasySSLContext();
105      }
106      return this.sslContext;
107  }
108  /**
109  * Returns the hashCode
110  * @return EasySSLProtocolSocketFactory.class.hashCode();
111  */
112 }
```

The screenshot shows the same Java code for the `Test.class` file, but with all annotations removed. The code is now presented in a more standard, unannotated form. The code is part of a class named `EasySSLProtocolSocketFactory`.

```
48  host, int port) throws IOException, UnknownHostException {
49      etFactory().createSocket(host, port);
50  }
51  /**
52   * #createSocket(java.lang.String,int,java.net.InetAddress,int)
53   */
54  host, int port, InetAddress clientHost, int clientPort) throws IOException, UnknownHostException {
55      etFactory().createSocket(host, port, clientHost, clientPort);
56  }
57  /**
58   * Connection to the given host within the given time limit.
59   * If older JREs that do not support connect timeout a controller thread is executed. The controller thread attempts to create a new socket within the given limit of time. If socket constructor does not return until the
60   * terminates and throws on @link ConnectTimeoutException
61   */
62  /**
63   * IP to bind the socket to
64   * machine
65   * @param params
66   *     Http connection parameters
67   * @throws IOException
68   *     while creating the socket
69   * @throws UnknownHostException
70   *     if the host cannot be determined
71   */
72  /**
73   * Returns the hashCode
74   * @return EasySSLProtocolSocketFactory.class.hashCode();
75   */
76  /**
77   * Creates a new SSLContext
78   * @return the SSLContext
79   */
80  private SSLContext createEasySSLContext() {
81      return SSLContext.getInstance("TLS");
82  }
83  /**
84   * Returns the hashCode
85   * @return EasySSLProtocolSocketFactory.class.hashCode();
86   */
87  public int hashCode() {
88      return EasySSLProtocolSocketFactory.class.hashCode();
89  }
90  /**
91   * Returns the hashCode
92   * @return EasySSLProtocolSocketFactory.class.hashCode();
93   */
94  public boolean equals(Object obj) {
95      return obj.getClass().equals(EasySSLProtocolSocketFactory.class);
96  }
97  /**
98   * Returns the hashCode
99   * @return EasySSLProtocolSocketFactory.class.hashCode();
100  */
101  public String toString() {
102      return "EasySSLProtocolSocketFactory";
103  }
104  /**
105   * Returns the hashCode
106   * @return EasySSLProtocolSocketFactory.class.hashCode();
107   */
108  public long hashCode() {
109      return EasySSLProtocolSocketFactory.class.hashCode();
110  }
111  /**
112   * Returns the hashCode
113   * @return EasySSLProtocolSocketFactory.class.hashCode();
114   */
115 }
```

The screenshot shows an IDE interface with the following details:

- Title Bar:** Quick Access, Java EE, Debug.
- Toolbar:** Standard IDE toolbar with various icons.
- Code Editor:** Displays the `EasyX509TrustManager.java` file. The code implements the `X509TrustManager` interface. It includes imports for Java security classes like `KeyStoreException`, `NoSuchAlgorithmException`, `TrustManagerFactory`, and `SSLContext`. The class has a constructor that initializes a `KeyStore` and a `TrustManagerFactory`. It overrides the `checkClientTrusted` and `checkServerTrusted` methods to accept self-signed certificates. A note at the top states: "This trust manager SHOULD NOT be used for productive systems due to security reasons, unless it is a conscious decision and you are perfectly aware of security implications of accepting self-signed certificates". Below this, there are links to authors and a disclaimer about active support.
- Status Bar:** Read-Only, Smart Insert, 1:1.

The screenshot shows an IDE interface with the following details:

- Title Bar:** Quick Access, Java EE, Debug.
- Toolbar:** Standard IDE toolbar with various icons.
- Code Editor:** Displays the `HttpMethod.java` file, which defines an enum for HTTP methods: GET, POST, PUT, DELETE, TRACE, CONNECT, and OPTIONS.
- Project Explorer:** Shows the project structure under `maulplatform-1.2.1.jar`. The structure includes packages for `com.ebay.maul.controller`, `com.ebay.maul.driver.api`, `com.ebay.maul.driver.db`, `com.ebay.maul.driver.mail`, `com.ebay.maul.driver.rdb`, `com.ebay.maul.driver.rss`, `com.ebay.maul.driver.web`, and `com.ebay.maul.driver.web.element`. It also lists various classes like `APIDriver.class`, `APIRequest.class`, `APIResponse.class`, `EasySSLProtocolSocketFactory.class`, `EasyX509TrustManager.class`, `HttpRequestFilter.class`, `ResponseFactory.class`, `RestClient.class`, `RestClientReportLogger.class`, `RESTResponse.class`, `ScreenShot.class`, `ScreenShotRemoteWebDriver.class`, `ScreenshotUtil.class`, `WebDriverConfig.class`, `WebDriverExceptionListener.class`, `WebDriverMode.class`, `WebUIXDriver.class`, `BaseHTMLPage.class`, `BasePageObject.class`, `Button.class`, `Checkbox.class`, `Form.class`, `GenericWebPage.class`, `HtmlElement.class`, `Image.class`, `IPage.class`, `Label.class`, `Link.class`, `LocatorSwitch.class`, and `LocatorType.class`.
- Outline View:** Shows the `HttpMethod` enum with its values: GET, POST, PUT, DELETE, TRACE, CONNECT, and OPTIONS.
- Status Bar:** Members calling constructors of 'WebPageSection' - in workspace.

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

- Project Explorer View:** Shows the project structure for "maulplatform-1.2.1.jar". The "RequestFilter.class" file is selected.
- Code Editor View:** Displays the Java code for the "RequestFilter" class, which extends "ClientFilter". The code includes imports for Jersey API classes and annotations like "@author chesnokamp".
- Quick Access Bar:** Located at the top right, it includes icons for Java EE, Debug, and other tools.
- Toolbars:** Standard Eclipse toolbars for file operations, search, and navigation.
- Right-hand View:** Shows the "Outl" view with the "RequestFilter" entry expanded, showing its methods: "RequestFilter()", "handle(ClientRequest)", and "handle(ClientResponse)".

```
1 package com.ebay.maul.driver.api;
2
3 import com.sun.jersey.api.client.ClientHandlerException;
4 import com.sun.jersey.api.client.ClientRequest;
5 import com.sun.jersey.api.client.ClientResponse;
6 import com.sun.jersey.api.client.filter.ClientFilter;
7
8 /**
9  * filter to inject the request object into the response object
10 */
11 @Author chesnokamp
12
13 public class RequestFilter extends ClientFilter {
14     @Override
15     public ClientResponse handle(ClientRequest cr) throws ClientHandlerException {
16         ClientResponse response = getNext().handle(cr);
17         response.getProperties().put("REQUEST", cr);
18         return response;
19     }
20 }
```

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

- Project Explorer:** Shows the project structure for "mauiplatform-1.2.1.jar".
- Editor:** Displays the Java code for the `ResponseFactory` class.
- Quick Access:** A toolbar at the top with various icons.
- Java EE:** A toolbar on the right.
- Outfit:** A toolbar on the far right.
- Members calling constructors of 'WebResponseSection' - in workspace:** A tooltip at the bottom of the editor area.

```
1 package com.ebay.maul.driver.api;
2
3 import java.util.Properties;
4
5 import javax.ws.rs.core.MediaType;
6
7 import com.sun.jersey.api.client.ClientResponse;
8 import com.sun.jersey.api.client.WebResource.Builder;
9
10 /**
11  * return proper response
12  * @author chasenkamp
13  */
14
15 public class Responsefactory {
16     public static ClientResponse getMethodTypeBuilder(RequestEntity requestEntity, HttpMethod httpMethod,
17             MediaType mediaType, Object requestEntityInfo, HttpMethod httpMethod,
18             Properties headers) {
19         Builder builder = requestBuilder;
20         if (headers != null) {
21             builder = addHeaders(builder, headers);
22         }
23         if (requestEntity != null) {
24             builder = builder.entity(requestEntity, mediaType);
25         }
26
27         switch (httpMethod) {
28             case GET:
29                 return builder.getClientResponse();
30             case POST:
31                 return builder.post(ClientResponse.class);
32             case DELETE:
33                 return builder.delete(ClientResponse.class);
34             case PUT:
35                 return builder.put(ClientResponse.class);
36             default:
37                 break;
38         }
39         throw new AssertionError("Http method " + httpMethod + " not supported");
40     }
41
42     private static Builder addHeaders(Builder builder, Properties headers) {
43         for (Object key : headers.keySet()) {
44             builder.header((String) key, headers.get(key));
45         }
46         return builder;
47     }
48 }
49
50 }
```

The screenshot shows the Java code for the `RestClient` class in an IDE. The code is annotated with Javadoc comments and imports various Jersey client-related classes. The `RestClient` class has a constructor that takes a `Class<T>` and a `Properties` object. It initializes a `ClientConfig` with `JSONConfiguration.FEATURE_POJO_MAPPING` and adds `APPLICATION_JSON_TYPE` to the accept header. It then creates a `Client` and a `RequestFilter`. The `entity` method sets the entity media type and the `postObject` method sends a POST request with the specified media type.

```
1 package com.ebay.maul.driver.api;
2
3 import javax.io.IOException;
4 import javax.ws.rs.client.Client;
5 import java.util.ArrayList;
6 import java.util.Arrays;
7 import java.util.HashSet;
8 import java.util.Properties;
9 import java.util.Set;
10
11 import javax.ws.rs.core.MediaType;
12 import javax.ws.rs.core.MultivaluedMap;
13
14 import org.codehaus.jackson.map.ObjectMapper;
15
16 import com.ebay.maul.helper.XMLHelper;
17 import com.google.common.base.Joiner;
18 import com.sun.jersey.api.client.Client;
19 import com.sun.jersey.api.client.ClientRequest;
20 import com.sun.jersey.api.client.ClientResponse;
21 import com.sun.jersey.api.client.UniformInterfaceException;
22 import com.sun.jersey.api.client.WebResource.Builder;
23 import com.sun.jersey.api.client.config.ClientConfig;
24 import com.sun.jersey.api.client.config.DefaultClientConfig;
25 import com.sun.jersey.api.json.JSONConfiguration;
26
27 /**
28 * RestClient is for RESTful service testing based on Jersey client
29 * @author ehsankmp
30 */
31 public class RestClient<T> {
32     private Client client;
33     private Builder builder;
34     private Class<T> model;
35
36     private MediaType mediatype = MediaType.APPLICATION_JSON_TYPE;
37     private Set<MediaType> accept = new HashSet<MediaType>();
38
39     private Object postObject;
40     private MediaType entityMediatype;
41
42     private T response;
43
44     private Properties headers;
45     private ClientResponse clientResponse;
46     private boolean continueOnBadRequest = false;
47
48     public RestClient(Class<T> model) {
49         init(model, null);
50     }
51
52     public RestClient(Class<T> model, Properties headers) {
53         init(model, headers);
54     }
55
56     private void init(Class<T> model, Properties headers) {
57         ClientConfig clientConfig = new DefaultClientConfig();
58         clientConfig.setFeatures().put(JSONConfiguration.FEATURE_POJO_MAPPING,
59             Boolean.TRUE);
60         this.accept.add(MediaType.APPLICATION_JSON_TYPE);
61         this.mediatype = model == null ? null : model.getContentType();
62         this.client = Client.create(clientConfig);
63         this.client.addFilter(new RequestFilter());
64         this.headers = headers;
65     }
66 }
```

This screenshot shows the same `RestClient` class code as the first one, but with additional methods and annotations. The `accept` method now takes a `MediaTypes.MediaType` parameter and adds it to the accept header. The `header` method sets a header with a value. The `entity` method sets the entity object and its media type. The `postObject` method sends a POST request with the specified media type and entity.

```
68     private void setAccept(MediaType type) {
69         this.builder = this.client.resource(url).getRequestBuilder();
70         this.builder.type(type);
71         this.builder.accept(this.accept.toArray(new MediaType[this.accept
72             .size()]));
73         this.builder.entity(this.postObject, this.entityMediatype);
74     }
75
76     /**
77      * Get the media type of the request
78      *
79      * @return media type
80      */
81     public MediaType getMediaTypeO {
82         return this.mediatype;
83     }
84
85     /**
86      * Set the media type of the request
87      *
88      * @param mediatype
89      * @param media type
90      * @return a rest client
91      */
92     public RestClient<T> accept(MediaType mediatype) {
93         this.mediatype = mediatype;
94         return this;
95     }
96
97     /**
98      * Set the accept types
99      *
100     * @param type
101     * @param media types
102     * @return a rest client
103     */
104     public RestClient<T> accept(MediaType... type) {
105         this.accept.addAll(new ArrayList<MediaType>(Arrays.asList(type)));
106         return this;
107     }
108
109     /**
110      * Set the headers
111      *
112      * @param name
113      * @param value
114      * @return value of the header
115      */
116     public RestClient<T> header(String name, Object value) {
117         if (this.headers == null)
118             this.headers = new Properties();
119         this.headers.put(name, value);
120         return this;
121     }
122
123     /**
124      * Set the request entity
125      *
126      * @param entity
127      * @param entity to be post
128      * @return a rest client
129      */
130     public RestClient<T> entity(Object entity, MediaType mediatype) {
131         this.postObject = entity;
132         this.entityMediatype = mediatype;
133         return this;
134     }
135 }
```

The screenshot shows a Java IDE interface with the RestClient class open in the editor. The code implements various HTTP methods (GET, POST, PUT, DELETE) using the Apache HttpClient library. It includes error handling for bad responses and logging of API steps.

```
133     return this;
134 }
135 /**
136  * Use to test negative cases like checking for client error handling of the
137  * service.
138 */
139 +
140     @return {@link RestClient}
141     public RestClient<T> continuedOnBadResponse() {
142         this.continuedOnBadResponse = true;
143         return this;
144     }
145 }
146 /**
147  * execute a GET method with the defined endpoint
148  */
149     public T get(URL url) {
150         return get(url, null);
151     }
152 }
153 /**
154  * execute a GET method with the defined endpoint and request object
155  */
156     public T get(URL url, Object entity) {
157         return doCall(url, entity, HttpMethod.GET);
158     }
159 }
160 /**
161  * execute a POST method with the defined endpoint
162  */
163     public T post(URL url) {
164         return post(url, null);
165     }
166 }
167 /**
168  * execute a POST method with the defined endpoint and request object
169  */
170     public T post(URL url, Object entity) {
171         return doCall(url, entity, HttpMethod.POST);
172     }
173 }
174 /**
175  * execute a PUT method with the defined endpoint
176  */
177     public T put(URL url) {
178         return put(url, null);
179     }
180 }
181 /**
182  * execute a PUT method with the defined endpoint and request object
183  */
184     public T put(URL url, Object entity) {
185         return doCall(url, entity, HttpMethod.PUT);
186     }
187 }
188 /**
189  * execute a DELETE method with the defined endpoint
190  */
191     public T delete(URL url) {
192         return delete(url, null);
193     }
194 }
195 /**
196  * execute a DELETE method with the defined endpoint and request object
197  */
198 }
```

The screenshot shows the same Java IDE interface with the RestClient class code. This version includes additional methods for handling client requests and responses, such as getClientRequest(), getClientResponse(), and logStatus(). It also includes suppression annotations (@SuppressWarnings("unchecked")) and exception handling logic.

```
197     * execute a DELETE method with the defined endpoint and request object
198     */
199     public T delete(URL url, Object entity) {
200         return doCall(url, entity, HttpMethod.DELETE);
201     }
202 }
203 /**
204  * getReturnResponse(ClientResponse response)
205  * this.response = response.getEntity(this.model);
206  * return this.response;
207 */
208 /**
209  * @return the response code, for example 200, or 401, or 500, etc.
210 */
211     public int getResponseCode() {
212         return this.clientResponse.getStatus();
213     }
214 /**
215  * private ClientResponse response(Object entity, HttpMethod httpMethod) {
216  *     this.postObject = entity;
217  *     this.clientResponse = ResponseFactory.getMethodType(this.builder,
218  *     this.url, this.userAgent, entity, httpMethod, this.headers);
219  *     return this.clientResponse;
220  */
221     @SuppressWarnings("unchecked")
222     private T doCall(URL url, Object entity, HttpMethod httpMethod) {
223         ClientIdentifier identifier = new ClientIdentifier(url);
224         ClientResponse clientResponse = response(entity, httpMethod);
225         logStatus(clientResponse);
226         if (clientResponse.equals(this.model)) {
227             try {
228                 this.response = (T) clientResponse;
229                 new RestClientReportLogger().logRestAPIStep(this);
230                 return (T) clientResponse;
231             } catch (UniformInterfaceException e) {
232                 return (T) getResponse();
233             }
234         } else {
235             if (continuedOnBadResponse) {
236                 terminatedOnBadResponse(clientResponse);
237             }
238             T response = getReturnResponse(clientResponse);
239             new RestClientReportLogger().logRestAPIStep(this);
240             return response;
241         }
242     }
243 }
244 /**
245  * public String getHttpMethod() {
246  *     return getClientRequest().getMethod();
247  */
248     private ClientRequest getClientRequest() {
249         return (ClientRequest) this.clientResponse.getProperties().get(
250             "REQUEST");
251     }
252 /**
253  * private void logStatus(ClientResponse response) {
254  *     System.out.println("Response getStatus:" + response.getStatus());
255  *     System.out.println("Response getCode:" + response.getStatusCode());
256  */
257     private void terminatedOnBadResponse(ClientResponse response) {
258         if (response.getStatus() < 200 || response.getStatus() > 299) {
259             System.out.println("Status: " + response.getStatus());
260             StringBuffer sb = getFailureMessage();
261         }
262 }
```

```
325
326 	/***
327 	* Convert the internal response entity to XML String. Note: Use this method
328 	* only when you are expecting on domain object, e.g. Item, User. If you
329 	* create
330 	* @link com.ebay.testinfrastructure.servicemodel.opidriver.RestClient)
331 	* using (code RestClient<String> client) It means that you expected a
332 	* String as the response, don't use this method otherwise it will convert
333 	* the entire response into a string, which is not what you need.
334 	* @return XML string serialized by the internal entity
335 	*/
336 	public String getResponseAsXml() {
337 		if (this.response == null)
338 			return null;
339
340 		if (this.response instanceof String)
341 			return (String) this.response;
342 		else {
343 			return XMLHelper.getXml(this.response);
344 		}
345 	}
346
347 	/***
348 	* Return the String representation of the Request
349 	* @return String representation of the posted object
350 	*/
351 	public String getRequestAsString() {
352 		if (this.postObject == null)
353 			return null;
354
355 		if (this.postObject instanceof String) {
356 			return (String) this.postObject;
357 		} else {
358 			return objectToJsonString(this.postObject);
359 		}
360 	}
361
362 	/***
363 	* Convert the Internal response entity to JSON String. Note: Use this
364 	* method only when you are expecting on domain object, e.g. Item, User. If
365 	* you create
366 	* @link com.ebay.testinfrastructure.servicemodel.opidriver.RestClient)
367 	* using (code RestClient<String> client) It means that you expected a
368 	* String as the response, don't use this method otherwise it will convert
369 	* the entire response into a JSON string, which is not what you need.
370 	* @return JSON string serialized by the internal entity
371 	*/
372 	public String getResponseAsJson() {
373 		if (this.response == null) {
374 			return null;
375 		}
376
377 		if (this.response instanceof String)
378 			return (String) this.response;
379 		else {
380 			return objectToJsonString(this.response);
381 		}
382 	}
383
384 	private String objectToJsonString(Object object) {
385 		ObjectMapper mapper = new ObjectMapper();
386 		try {
387 			return mapper.writerWithDefaultPrettyPrinter().writeValueAsString(
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
647
648
649
649
650
651
652
653
654
655
656
656
657
658
658
659
659
660
661
662
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1530
1531
1531
1532
1532
1533
1533
1534
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1540
1541
1541
1542
1542
1543
1543
1544
1544
1545
1545
1546
1546
1547
1547
1548
1548
1549
1549
1550
1550
1551
1551
1552
1552
1553
1553
1554
1554
1555
1555
1556
1556
1557
1557
1558
1558
1559
1559
1560
1560
1561
1561
1562
1562
1563
1563
1564
1564
1565
1565
1566
1566
1567
1567
1568
1568
1569
1569
1570
1570
1571
1571
1572
1572
1573
1573
1574
1574
1575
1575
1576
1576
1577
1577
1578
1578
1579
1579
1580
1580
1581
1581
1582
1582
1583
1583
1584
1584
1585
1585
1586
1586
1587
1587
1588
1588
1589
1589
1590
1590
1591
1591
1592
1592
1593
1593
1594
1594
1595
1595
1596
1596
1597
1597
1598
1598
1599
1599
1600
1600
1601
1601
1602
1602
1603
1603
1604
1604
1605
1605
1606
1606
1607
1607
1608
1608
1609
1609
1610
1610
1611
1611
1612
1612
1613
1613
1614
1614
1615
1615
1616
1616
1617
1617
1618
1618
1619
1619
1620
1620
1621
1621
1622
1622
1623
1623
1624
1624
1625
1625
1626
1626
1627
1627
1628
1628
1629
1629
1630
1630
1631
1631
1632
1632
1633
1633
1634
1634
1635
1635
1636
1636
1637
1637
1638
1638
1639
1639
1640
1640
1641
1641
1642
1642
1643
1643
1644
1644
1645
1645
1646
1646
1647
1647
1648
1648
1649
1649
1650
1650
1651
1651
1652
1652
1653
1653
1654
1654
1655
1655
1656
1656
1657
1657
1658
1658
1659
1659
1660
1660
1661
1661
1662
1662
1663
1663
1664
1664
1665
1665
1666
1666
1667
1667
1668
1668
1669
1669
1670
1670
1671
1671
1672
1672
1673
1673
1674
1674
1675
1675
1676
1676
1677
1677
1678
1678
1679
1679
1680
1680
1681
1681
1682
1682
1683
1683
1684
1684
1685
1685
1686
1686
1687
1687
1688
1688
1689
1689
1690
1690
1691
1691
1692
1692
1693
1693
1694
1694
1695
1695
1696
1696
1697
1697
1698
1698
1699
1699
1700
1700
1701
1701
1702
1702
1703
1703
1704
1704
1705
1705
1706
1706
1707
1707
1708
1708
1709
1709
1710
1710
1711
1711
1712
1712
1713
1713
1714
1714
1715
1715
1716
1716
1717
1717
1718
1718
1719
1719
1720
1720
1721
1721
1722
1722
1723
1723
1724
1724
1725
1725
1726
1726
1727
1727
1728
1728
1729
1729
1730
1730
1731
1731
1732
1732
1733
1733
1734
1734
1735
1735
1736
1736
1737
1737
1738
1738
1739
1739
1740
1740
1741
1741
1742
1742
1743
1743
1744
1744
1745
1745
1746
1746
1747
1747
1748
1748
1749
1749
1750
1750
1751
1751
1752
1752
1753
1753
1754
1754
1755
1755
1756
1756
1757
1757
1758
1758
1759
1759
1760
1760
1761
1761
1762
1762
1763
1763
1764
1764
1765
1765
1766
1766
1767
1767
1768
1768
1769
1769
1770
1770
1771
1771
1772
1772
1773
1773
1774
1774
1775
1775
1776
1776
1777
1777
1778
1778
1779

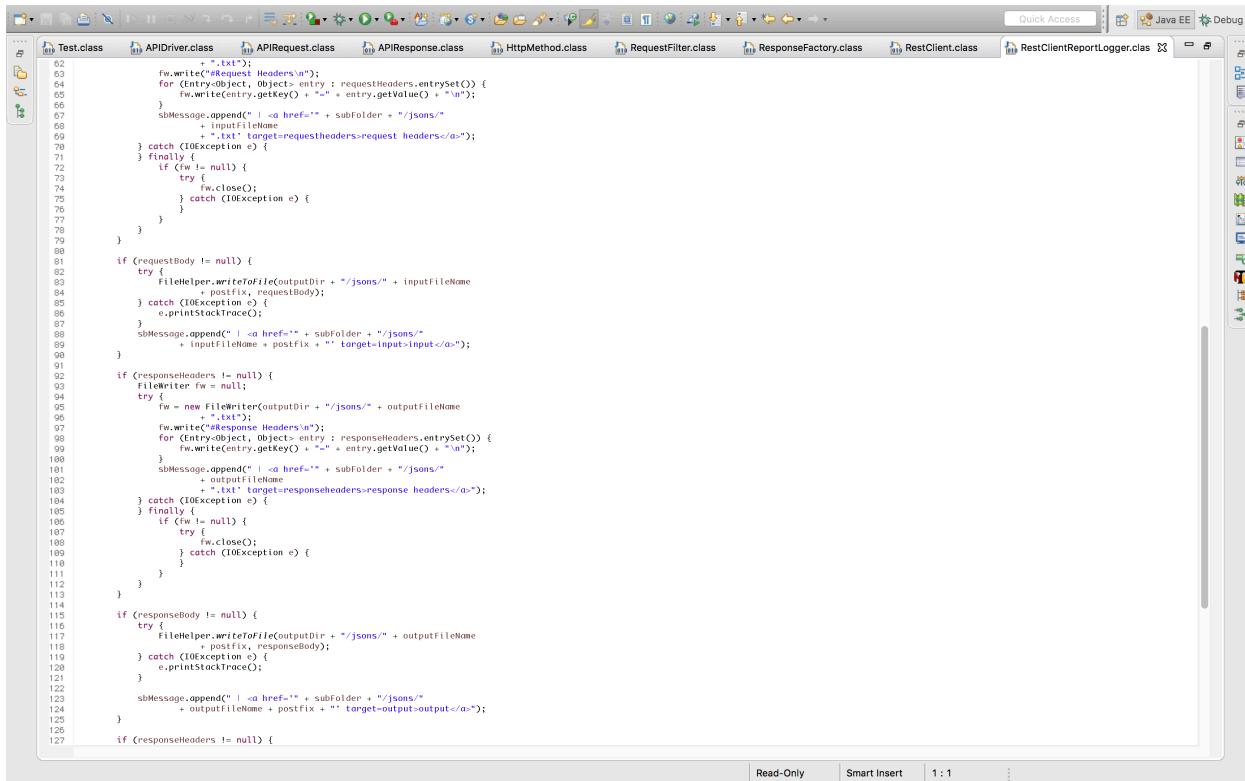
```

The screenshot shows the Eclipse IDE interface with the RestClient class selected in the left navigation bar. The main editor window displays the following Java code:

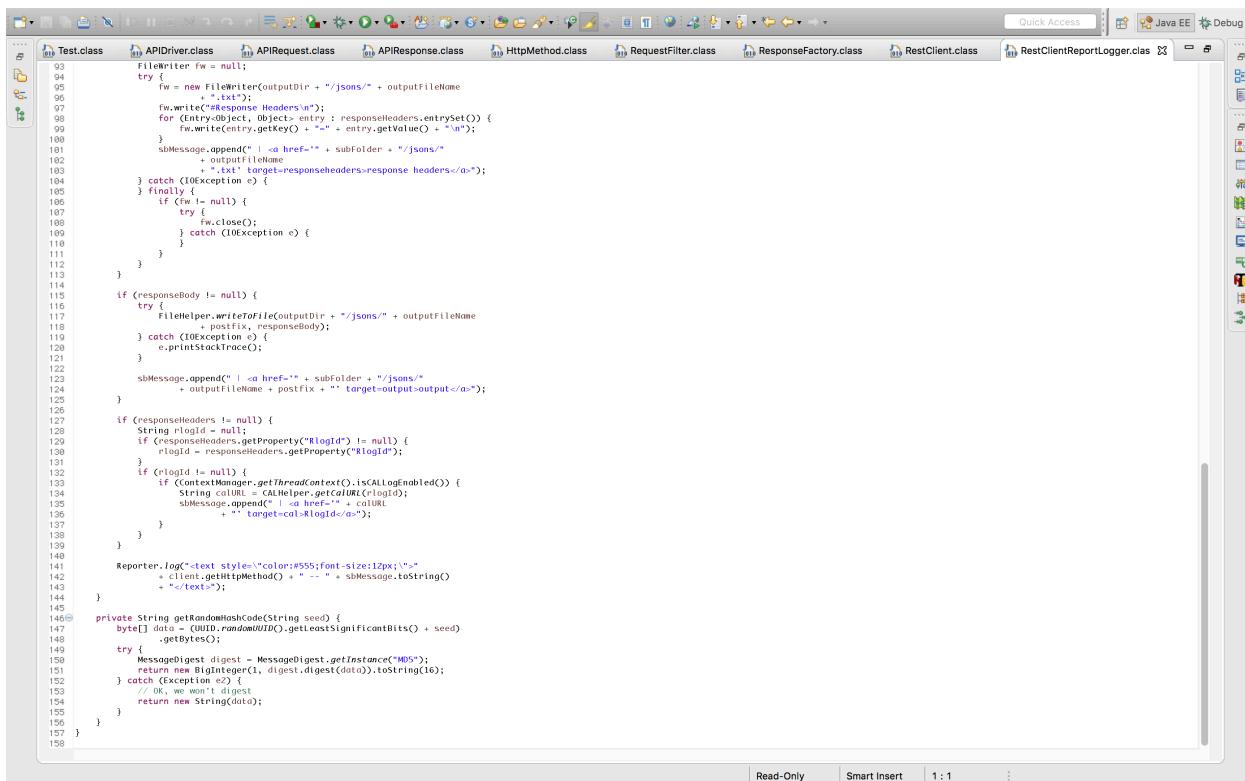
```
1 package com.ebay.moui.driver.api;
2
3 import java.io.File;
4 import java.io.IOException;
5 import java.io.InputStream;
6 import java.math.BigInteger;
7 import java.security.MessageDigest;
8 import java.util.Properties;
9 import java.util.UUID;
10
11 import javax.ws.rs.core.MediaType;
12
13 import org.junit.Test;
14
15 import com.ebay.moui.controller.ContextManager;
16 import com.ebay.moui.helper.CALHelper;
17 import com.ebay.moui.helper.FileHelper;
18
19 /**
20 * This class helps log RESTful service steps on TestNG report
21 */
22
23 * Author bshen (Binfeng, Shen)
24 */
25 public class RestClientReportLogger {
26
27     /**
28      * Log RESTful service steps with web resource URI, request headers, request
29      * body and response body
30      *
31      * @param client
32      * @return
33     */
34     public void logRestAPIStep(RestClient<?> client) {
35         Properties requestHeaders = client.getRequestHeaders();
36         String requestBody = client.getRequestBody() != null ? client.getRequest()
37             .toString() : null;
38         Properties responseHeaders = client.getResponseHeaders();
39         String responseBody = client.getResponse() != null ? client
40             .getResponseBody().toString() : null;
41
42         StringBuffer sbMessage = new StringBuffer(<>);
43         sbMessage.append("<a href=\"" + client.getEndPoint() +
44             " target=_apiurl></a>");
45
46         String inputFileName = getRequestBodyName("rest-input");
47         String outputFileName = getResponseBodyName("rest-output");
48         String subFolder = ContextManager.getGlobalContext().getTestNGContext()
49             .getSuite().getName();
50         String outputDir = ContextManager.getGlobalContext()
51             .getOutputDir();
52         new File(outputDir + "/" + subFolder).mkdirs();
53
54         String postfix = ".json";
55         if (client.getMediaType().equals(MediaType.APPLICATION_XML_TYPE)) {
56             postfix = ".xml";
57         }
58
59         if (requestHeaders != null) {
60             FileWriter fw = null;
61             try {
62                 fw = new FileWriter(outputDir + "/jsons/" + inputFileName
63                     + ".txt");
64                 fw.write("#Request Headers\n");
65                 for (Entry<Object, Object> entry : requestHeaders.entrySet()) {
66                     fw.write(entry.getKey() + "=" + entry.getValue() + "\n");
67                 }
68             } catch (IOException e) {
69                 e.printStackTrace();
70             } finally {
71                 if (fw != null) {
72                     fw.close();
73                 }
74             }
75         }
76     }
77 }
```

The screenshot shows the Eclipse IDE interface with the RestClientReportLogger class selected in the left navigation bar. The main editor window displays the following Java code:

```
1 package com.ebay.moui.driver.api;
2
3 import java.io.File;
4 import java.io.FileWriter;
5 import java.io.IOException;
6 import java.math.BigInteger;
7 import java.security.MessageDigest;
8 import java.util.Properties;
9 import java.util.UUID;
10
11 import javax.ws.rs.core.MediaType;
12
13 import org.junit.Test;
14
15 import com.ebay.moui.controller.ContextManager;
16 import com.ebay.moui.helper.CALHelper;
17 import com.ebay.moui.helper.FileHelper;
18
19 /**
20 * This class helps log RESTful service steps on TestNG report
21 */
22
23 * Author bshen (Binfeng, Shen)
24 */
25 public class RestClientReportLogger {
26
27     /**
28      * Log RESTful service steps with web resource URI, request headers, request
29      * body and response body
30      *
31      * @param client
32      * @return
33     */
34     public void logRestAPIStep(RestClient<?> client) {
35         Properties requestHeaders = client.getRequestHeaders();
36         String requestBody = client.getRequestBody() != null ? client.getRequest()
37             .toString() : null;
38         Properties responseHeaders = client.getResponseHeaders();
39         String responseBody = client.getResponse() != null ? client
40             .getResponseBody().toString() : null;
41
42         StringBuffer sbMessage = new StringBuffer(<>);
43         sbMessage.append("<a href=\"" + client.getEndPoint() +
44             " target=_apiurl></a>");
45
46         String inputFileName = getRequestBodyName("rest-input");
47         String outputFileName = getResponseBodyName("rest-output");
48         String subFolder = ContextManager.getGlobalContext().getTestNGContext()
49             .getSuite().getName();
50         String outputDir = ContextManager.getGlobalContext()
51             .getOutputDir();
52         new File(outputDir + "/" + subFolder).mkdirs();
53
54         String postfix = ".json";
55         if (client.getMediaType().equals(MediaType.APPLICATION_XML_TYPE)) {
56             postfix = ".xml";
57         }
58
59         if (requestHeaders != null) {
60             FileWriter fw = null;
61             try {
62                 fw = new FileWriter(outputDir + "/jsons/" + inputFileName
63                     + ".txt");
64                 fw.write("#Request Headers\n");
65                 for (Entry<Object, Object> entry : requestHeaders.entrySet()) {
66                     fw.write(entry.getKey() + "=" + entry.getValue() + "\n");
67                 }
68             } catch (IOException e) {
69                 e.printStackTrace();
70             } finally {
71                 if (fw != null) {
72                     fw.close();
73                 }
74             }
75         }
76     }
77 }
```



```
52     fw.write("<?xml");
53     fw.write("<request Headers=>");
54     for (Entry<Object, Object> entry : requestHeaders.entrySet()) {
55         fw.write(entry.getKey() + "=" + entry.getValue() + "\n");
56     }
57     sbMessage.append(" | <a href=\"" + subFolder + "/jsons/" +
58                     + inputFileName +
59                     + "> " + entry.getKey() + "=" + entry.getValue() + "</a>\"");
60     } catch (IOException e) {
61         e.printStackTrace();
62     } finally {
63         if (fw != null) {
64             try {
65                 fw.close();
66             } catch (IOException e) {
67             }
68         }
69     }
70 }
71 if (requestBody != null) {
72     try {
73         FileHelper.writeFile(outputDir + "/jsons/" + inputFileName
74                         + postfix, requestBody);
75     } catch (IOException e) {
76         e.printStackTrace();
77     }
78 }
79 sbMessage.append(" | <a href=\"" + subFolder + "/jsons/" +
80                     + inputFileName + postfix + " target=<input type=" +
81                     + "text" + " target=>requestheaders:request headers</a>\"");
82 }
83 if (responseHeaders != null) {
84     FileWriter fw = new FileWriter(outputDir + "/jsons/" + outputFileName
85                                 + ".txt");
86     fw.write("<response Headers=>");
87     for (Entry<Object, Object> entry : responseHeaders.entrySet()) {
88         fw.write(entry.getKey() + "=" + entry.getValue() + "\n");
89     }
90     sbMessage.append(" | <a href=\"" + subFolder + "/jsons/" +
91                     + outputFileName +
92                     + "> " + entry.getKey() + "=" + entry.getValue() + "</a>\"");
93     } catch (IOException e) {
94     } finally {
95         if (fw != null) {
96             try {
97                 fw.close();
98             } catch (IOException e) {
99             }
100 }
101 }
102 if (responseBody != null) {
103     try {
104         FileHelper.writeFile(outputDir + "/jsons/" + outputFileName
105                         + postfix, responseBody);
106     } catch (IOException e) {
107         e.printStackTrace();
108     }
109 }
110 sbMessage.append(" | <a href=\"" + subFolder + "/jsons/" +
111                     + outputFileName + postfix + " target=<output type=" +
112                     + "text" + " target=>responseheaders:response headers</a>\"");
113 }
114 if (responseHeaders != null) {
115     try {
116         FileHelper.writeFile(outputDir + "/jsons/" + outputFileName
117                         + postfix, responseBody);
118     } catch (IOException e) {
119         e.printStackTrace();
120     }
121 }
122 sbMessage.append(" | <a href=\"" + subFolder + "/jsons/" +
123                     + outputFileName + postfix + " target=<output type=" +
124                     + "text" + " target=>responseheaders:response headers</a>\"");
125 }
126 if (responseHeaders != null) {
```



```
93     FileWriter fw = null;
94     try {
95         fw = new FileWriter(outputDir + "/jsons/" + outputFileName
96                         + ".txt");
97     } catch (IOException e) {
98     }
99     fw.write("<response Headers=>");
100    for (Entry<Object, Object> entry : responseHeaders.entrySet()) {
101        fw.write(entry.getKey() + "=" + entry.getValue() + "\n");
102    }
103    sbMessage.append(" | <a href=\"" + subFolder + "/jsons/" +
104                     + outputFileName +
105                     + "> " + entry.getKey() + "=" + entry.getValue() + "</a>\"");
106    } catch (IOException e) {
107    } finally {
108        if (fw != null) {
109            try {
110                fw.close();
111            } catch (IOException e) {
112            }
113        }
114    }
115    if (responseBody != null) {
116        try {
117            FileHelper.writeFile(outputDir + "/jsons/" + outputFileName
118                            + postfix, responseBody);
119        } catch (IOException e) {
120            e.printStackTrace();
121        }
122    }
123    sbMessage.append(" | <a href=\"" + subFolder + "/jsons/" +
124                     + outputFileName + postfix + " target=<output type=" +
125                     + "text" + " target=>responseheaders:response headers</a>\"");
126 }
127 if (responseHeaders != null) {
128     String logId = null;
129     if (responseHeaders.getProperty("logId") != null) {
130         logId = responseHeaders.getProperty("logId");
131     }
132     if (logId != null) {
133         if (ContextManager.getThreadContext().isALogEnabled()) {
134             String callURL = CallHelper.getCallURL(logId);
135             sbMessage.append(" | <a href=\"" + callURL +
136                             + "> " + entry.getKey() + "=" + entry.getValue() + "</a>\"");
137         }
138     }
139 }
140 Reporter.log("<text style='color:#555;font-size:12px;'>" +
141             + client.getHttpMethod() + " -- " + sbMessage.toString() +
142             + "</text>");
143 }
144 }
145 }
146 private String getandomDigestCode(String seed) {
147     byte[] digest = MD5RandomDDigest.getLeastSignificantBits() + seed
148         .getBytes();
149     try {
150         MessageDigest digest = MessageDigest.getInstance("MD5");
151         return new BigInteger(1, digest.digest(digest)).toString(16);
152     } catch (Exception e) {
153         // OK, we won't digest
154         return new String(digest);
155     }
156 }
157 }
```

The screenshot shows the Eclipse IDE interface with the Java EE perspective selected. The central view displays the code for the `RESTResponse` class. The code is as follows:

```
package com.ebay.mau.driver.api;
import java.util.HashMap;
import java.util.Map;
public class RESTResponse {
    private String responseBody;
    private int status;
    private Map<String, String> statusInfo;
    private Map<String, String> headers = new HashMap<String, String>();
    public String getResponseBody() {
        return responseBody;
    }
    public void setResponseBody(String responseBody) {
        this.responseBody = responseBody;
    }
    public int getStatus() {
        return status;
    }
    public void setStatus(int status) {
        this.status = status;
    }
    public String getStatusInfo() {
        return statusInfo;
    }
    public void setStatusInfo(String statusInfo) {
        this.statusInfo = statusInfo;
    }
    public Map<String, String> getHeaders() {
        return headers;
    }
    public void setHeaders(Map<String, String> headers) {
        this.headers = headers;
    }
    @Override
    public String toString() {
        return "RESTResponse{" + "responseBody=" + responseBody + "\\"+
               ", status=" + status + ", statusInfo=" + statusInfo + "\\"+
               ", headers=" + headers + "}";
    }
}
```

To the right of the code, a UML class diagram for `RESTResponse` is shown. The class has the following attributes and methods:

- Attributes:**
  - `responseBody : String`
  - `status : int`
  - `statusInfo : String`
  - `headers : Map<String, String>`
- Operations:**
  - `getResponseBody() : String`
  - `setStatus(int) : void`
  - `setStatusInfo(String) : void`
  - `getHeaders() : Map<String, String>`
  - `setHeaders(Map<String, String>) : void`
  - `toString() : String`

At the bottom of the code editor, there is a toolbar with various icons for navigating and managing code.

## DB Driver:

The screenshot shows an IDE interface with multiple tabs open. The tabs include ResponseFactory.class, RestClient.class, RestClientReportLogger.class, RESTResponse.class, DBDriver.class, and MongoDBDriver.class. The code visible in the main editor is for the DBDriver class, which extends the MongoClient class. It includes imports for various Java packages such as java.io, java.util, and org.mongodb.driver. The class defines a static inner class DBHost and two static methods: DBHost(String name, String user, String password, String url) and DBHost(String name, String user, String password, String sid, String hostAddress, String port). The code uses reflection to set properties like driver and url based on the provided parameters.

```
1 package com.ebay.moui.driver.db;
2
3 import java.io.BufferedReader;
4 import java.io.BufferedInputStream;
5 import java.io.IOException;
6 import java.io.InputStreamReader;
7 import java.io.OutputStream;
8 import java.sql.ResultSet;
9 import java.sql.ResultSetMetaData;
10 import java.sql.SQLException;
11 import java.util.ArrayList;
12 import java.util.HashMap;
13 import java.util.Iterator;
14 import java.util.List;
15 import java.util.Map;
16 import java.util.Properties;
17 import java.util.Stack;
18
19 import org.apache.log4j.Logger;
20 import org.sequelize.model.SequelizeResultMapper;
21 import org.sequelize.session.SequelizeException;
22 import org.sequelize.session.SequelizeFinderException;
23 import org.sequelize.session.Session;
24 import org.sequelize.session.SessionFactory;
25 import org.sequelize.utils.DAOUtils;
26 import org.sequelize.utils.Utils;
27
28 import com.ebay.moui.controller.Context;
29 import com.ebay.moui.controller.ContextManager;
30 import com.ebay.moui.util.logging.*;
31 import com.ebay.moui.exception.DBException;
32 import com.ebay.moui.exception.DBFinderException;
33 import com.ebay.moui.helper.StringHelper;
34
35 /**
36  * DBDriver is based on sequelize and provides ways for SQL based DB testing.
37  */
38
39 public class DBDriver {
40     public static class DBHost {
41         String name;
42         String driver;
43         String password;
44         String sid;
45         String hostAddress;
46         String port;
47         String driver = "oracle.jdbc.driver.OracleDriver";
48         String url;
49
50         public DBHost(String name, String user, String password, String url,
51             String driver) {
52             this.name = name;
53             this.user = user;
54             this.password = password;
55             this.url = url;
56             this.driver = driver;
57         }
58
59         public DBHost(String name, String user, String password, String sid,
60             String hostAddress, String port) {
61             this.name = name;
62             this.user = user;
63             this.password = password;
64             this.sid = sid;
65             this.hostAddress = hostAddress;
66             this.port = port;
67         }
68     }
69 }
```

The screenshot shows an IDE interface with multiple tabs open. The active tab is 'ResponseFactory.class'. The code in the editor is as follows:

```
57
58
59     public DBHost(String name, String user, String password, String sid,
60                  String hostAddress, String port, String driver) {
61         this.name = name;
62         this.user = user;
63         this.password = password;
64         this.sid = sid;
65         this.hostAddress = hostAddress;
66         this.port = port;
67         this.driver = driver;
68     }
69
70     public String getDriver() {
71         return driver;
72     }
73
74     public String getHostAddress() {
75         return hostAddress;
76     }
77
78     public String getName() {
79         return name;
80     }
81
82     public String getPassword() {
83         return password;
84     }
85
86     public String getPort() {
87         return port;
88     }
89
90     public String getId() {
91         return sid;
92     }
93
94     public String getUrl() {
95         return url;
96     }
97
98     public String getUser() {
99         return user;
100    }
101}
102
103 protected static final Logger logger = Logging.getLogger(DBDriver.class);
104
105 private static SessionFactory factory = null;
106 private static ThreadLocal<Map<String, Session>> threadLocalSession = new ThreadLocal<Map<String, Session>>();
107
108 private static ThreadLocal<Stack<DBHost>> threadLocalDBHostStack = new ThreadLocal<Stack<DBHost>>();
109
110 static {
111     initSessionPools();
112 }
113
114 /**
115  * Cleanup Sessions for the thread
116  */
117 public static void cleanup() {
118     Map<String, Sessions> map = threadLocalSession.get();
119     if (map != null) {
120         try {
121             if (ContextManager.getThreadContext() != null)
122                 logger.info(*n*
```

This screenshot shows the RESTResponse class code in an IDE. The code is annotated with numerous comments explaining the logic. It includes methods for creating DBHost objects, initializing pools, and managing sessions. The code uses properties like dbHostHome, user, password, and driverClass to configure the database connection.

```
131     if (ContextManager.getThreadContext() != null)
132         logger.info("n" + ContextManager.getThreadContext()
133                     .getTestMethodSignature()
134                     + " - [Cleaning up DB Sessions : " + map);
135     for (Session session : map.values()) {
136         session.close();
137     }
138 }
139 } catch (SQLException e) {
140     // ignore
141 }
142 threadLocalDBHostStack.set(null);
143 threadLocalSession.set(null);
144 }
145 }
146 /**
147 * Create your own DBHost. Useful if you are not planning to connect to the
148 * standard schema sets
149 */
150 @Param(dbHostHome
151 @Param user
152 @Param password
153 @Param driverClass
154 @Param driver
155 @Param driverClass
156 @Param
157 public static DBHost createDBHost(String dbHostHome, String user,
158                                     String password, String jdbcUrl, String driverClass) {
159     return new DBHost(dbHostHome, user, password, jdbcUrl, driverClass);
160 }
161 }
162 /**
163 * initSessionPools()
164 */
165 Properties prop = new Properties();
166 prop.setProperty("log4j.rootLogger.level", "INFO");
167 prop.setProperty("log4j.logger.level.sql", "CONFIG");
168 // List<DBHost> hostList = retrieveHosts();
169 // for (DBHost hostInfo : hostList) {
170 // loadHostConfig(prop, hostInfo);
171 // }
172 SessionFactory.forceOrMappingCreation();
173 factory = SessionFactory.getInstance(prop);
174 }
175 /**
176 * public DBDriver() {
177 }
178 */
179 private Session getSession(DBHost host) throws SQLException {
180     if (threadLocalSession.get() == null) {
181         try {
182             threadLocalSession.get().put(host.getNameO);
183         } catch (SQLException e) {
184             factory.createSession(host.getNameO);
185         }
186         Properties prop = new Properties();
187         loadHostConfig(prop, host); // Write DBHost information to
188         // properties
189         SessionFactory.getInstance(prop); // This will load the DB
190         // information
191         threadLocalSession.get().put(host.getNameO,
192                                     factory.createSession(host.getNameO));
193     }
194 }
195 }
```

This screenshot continues the RESTResponse class code. It includes methods for writing content to a file and closing the session. The code handles various exceptions and ensures proper cleanup of resources.

```
196     if (threadLocalDBHostStack.get() == null) {
197         threadLocalDBHostStack.set(new Stack<DBHost>());
198     }
199     threadLocalDBHostStack.get().push(host);
200 }
201 return threadLocalSession.get().getHost().getName();
202 }
203 */
204 /**
205 * Saves HTML Source
206 */
207 @Param(path
208 @Param selenium
209 @Param
210 */
211 private void writeToFile(String path, String content) throws IOException {
212     FileOutputStream fos = null;
213     OutputStreamWriter osw = null;
214     BufferedWriter bw = null;
215     try {
216         fos = new FileOutputStream(path);
217         osw = new OutputStreamWriter(fos, "UTF8");
218         bw = new BufferedWriter(osw);
219         bw.write(content);
220     } finally {
221         if (bw != null) {
222             try {
223                 bw.close();
224             } catch (Exception e) {
225             } // KEEPME
226         }
227         if (osw != null) {
228             try {
229                 osw.close();
230             } catch (Exception e) {
231             } // KEEPME
232         }
233         if (fos != null) {
234             try {
235                 fos.close();
236             } catch (Exception e) {
237             } // KEEPME
238     }
239 }
240 */
241 /**
242 * Cleanup last DB Session for the thread
243 */
244 public void close() throws DBException {
245     /*
246     * If (threadLocalDBHostStack.get() == null || 
247     * threadLocalDBHostStack.get().isEmpty()) throw new
248     * DBException("You haven't executed a query on any DBHosts yet.");
249     */
250     if (threadLocalDBHostStack.get() != null
251         && !threadLocalDBHostStack.get().isEmpty())
252         close(threadLocalDBHostStack.get().pop());
253 }
254 */
255 }
```

```

263     * Close the DB Connection For particular DBHost. ECAT automatically closes
264     * all connections after test case execution is complete. So this is only
265     * useful if you are opening more than 10 connections per execution.
266     *
267     * @param host
268     * @throws DBException
269
270     public void closeDBHost(DBHost host) throws DBException {
271         try {
272             if (threadLocalSession.get() != null
273                 && threadLocalSession.get().containsKey(host.getName())) {
274                 Session session = threadLocalSession.get().remove(
275                     host.getName());
276                 session.close();
277             }
278
279             if (threadLocalDBHostStack.get() != null
280                 && threadLocalDBHostStack.get().contains(host)) {
281                 threadLocalDBHostStack.get().remove(host);
282             }
283         } catch (SequoniteException e) {
284             throw new DBException(e);
285         }
286     }
287
288     /**
289      * Commits to the DBHost of current thread.
290      *
291      * @throws DBException
292
293     public void commit() throws DBException {
294         /*
295          * If (threadLocalDBHostStack.get() == null || 
296          * threadLocalDBHostStack.get().isEmpty()) throw new 
297          * DBException("You haven't executed a query on any DBHosts yet.");
298         */
299
300         if (threadLocalDBHostStack.get() != null
301             && !threadLocalDBHostStack.get().isEmpty())
302             commit(threadLocalDBHostStack.get().peek());
303     }
304
305     /**
306      * Commits to the DBHost.
307      *
308      * @param host
309      * @throws DBException
310      */
311
312     public void commit(DBHost host) throws DBException {
313         try {
314             Session session = getSession(host);
315             session.commit();
316         } catch (SequoniteException e) {
317             throw new DBException(e);
318         }
319     }
320
321     /**
322      * Executes a database stored procedure and a) populates the OUT parameters
323      * into inParams b) if the stored procedure returns resultsets, this method
324      * will return them Assumption: the Parameters Order is IN first and then
325      * the OUT, the exception being for functions where they have a return
326      * value. So the first parameter will be a OUT parameter.
327      *
328      * Example: <code>
329      * </pre>
330      */
331
332     @Preconditions(protocols)
333     public List<Object> executeCall(DBHost host, String procSQL, Object[] inParams,
334                                     List<String> outParams) throws DBException {
335
336         String query = procSQL.replaceAll("\\?", "?");
337         if (!query.startsWith("SELECT")) {
338             throw new DBException(
339                 "executeCall() is used to execute stored procedures / Functions.");
340         }
341
342         @SuppressWarnings("rawtypes")
343         public List<Object> executeCall(DBHost host, String procSQL, Object[] inParams,
344                                         List<String> outParams) throws DBException {
345
346             try {
347                 String query = procSQL.replaceAll("\\?", "?");
348                 if (!query.startsWith("SELECT")) {
349                     if (query.startsWith("UPDATE")) {
350                         throw new DBException(
351                             "executeAll() is used to execute stored procedures / Functions.");
352
353                     Session session = getSession(host);
354                     List<Object> result = session.executeCall(procSQL, inParams, outParams);
355
356                     Logging.logInfo("Host=" + host.getName() + ", procSQL=" +
357                         procSQL + ", inParams=" + Utils.dumpArray(inParams) +
358                         ", outParams=" + outParams.toString() + ", false");
359
360                     return result;
361                 } catch (SequoniteException e) {
362                     Logging.logInfo("Host=" + host.getName() + ", procSQL=" +
363                         procSQL + ", inParams=" + Utils.dumpArray(inParams) +
364                         ", true");
365
366                     throw new DBException(e);
367                 }
368             }
369
370             /*
371              * Execute a query and returns an iterator with results, only first 50 rows
372              * returned. It can also populate an array list with column names. This is
373              * convenient to be used by test author. Example: <code>
374              * <pre>
375              * List<String> columnNamesList = new ArrayList<String>();
376              * Iterator<String> resultsIterator = executeQuery(DBHosts.StagingDB.UserHostReadPhysical, "SELECT ID, NAME FROM EMP_TABLE", columnNamesList);
377              * String[] rowdata = resultsIterator.next();
378              * String id = rowdata[columnNamesList.indexOf("ID")]; //Retrieve the id & name for the first row in the array
379              * String name = rowdata[columnNamesList.indexOf("NAME")];
380              * </pre>
381              */
382
383             #param host
384             #param query
385             #param columnNamesList
386             #return
387             #throws DBFinderException
388             #throws Exception
389
390             public Iterator<String> executeQuery(DBHost host, String query,
391                 List<String> columnNamesList) throws DBException, DBFinderException {

```

```

326     *
327     * Example: <code>
328     * </pre>
329     * List result = executeCall(DBHosts.StagingDB.UserHostWritePhysical,
330                               "call reviseTotal(?,?)",
331                               new Object[]{"John", 1234}, new ArrayList());
332
333     </pre>
334     </code>
335
336     #param host
337     #param procSQL
338     #param inParams
339     #param outParams
340
341     #return
342     #throws DBException
343
344     @SuppressWarnings("rawtypes")
345     public List<Object> executeCall(DBHost host, String procSQL, Object[] inParams,
346                                     List<String> outParams) throws DBException {
347
348         try {
349             String query = procSQL.replaceAll("\\?", "?");
350             if (!query.startsWith("SELECT")) {
351                 if (query.startsWith("UPDATE")) {
352                     throw new DBException(
353                         "executeAll() is used to execute stored procedures / Functions.");
354
355                     Session session = getSession(host);
356                     List<Object> result = session.executeCall(procSQL, inParams, outParams);
357
358                     Logging.logInfo("Host=" + host.getName() + ", procSQL=" +
359                         procSQL + ", inParams=" + Utils.dumpArray(inParams) +
360                         ", outParams=" + outParams.toString() + ", false");
361
362                     return result;
363                 } catch (SequoniteException e) {
364                     Logging.logInfo("Host=" + host.getName() + ", procSQL=" +
365                         procSQL + ", inParams=" + Utils.dumpArray(inParams) +
366                         ", true");
367
368                     throw new DBException(e);
369                 }
370             }
371
372             /*
373              * Execute a query and returns an iterator with results, only first 50 rows
374              * returned. It can also populate an array list with column names. This is
375              * convenient to be used by test author. Example: <code>
376              * <pre>
377              * List<String> columnNamesList = new ArrayList<String>();
378              * Iterator<String> resultsIterator = executeQuery(DBHosts.StagingDB.UserHostReadPhysical, "SELECT ID, NAME FROM EMP_TABLE", columnNamesList);
379              * String[] rowdata = resultsIterator.next();
380              * String id = rowdata[columnNamesList.indexOf("ID")]; //Retrieve the id & name for the first row in the array
381              * String name = rowdata[columnNamesList.indexOf("NAME")];
382              * </pre>
383              */
384
385             #param host
386             #param query
387             #param columnNamesList
388             #return
389             #throws DBFinderException
390             #throws Exception
391
392             public Iterator<String> executeQuery(DBHost host, String query,
393                 List<String> columnNamesList) throws DBException, DBFinderException {

```

```

398     * Execute a query and returns an iterator with results, rows returned as
399     * array and rowsPerPage specified. It can also populate an array list
400     * with column names. This is convenient to be used by test author. Example:
401     * <code>
402     * List<String> columnNameList = new ArrayList<String>();
403     * Iterator<String> resultsIterator = executeQuery(DB2Hosts.StagingDB.UserHostReadPhysical,
404     * "SELECT * FROM EMP_TABLE WHERE ID=? AND NAME=?",
405     * new Object[]{12345, "John"}, 1, 50, columnNameList);
406     *
407     * String[] rowdata = resultsIterator.next();
408     * String id = rowdata[columnNameList.indexOf("ID")]; //Retrieve the id & name for the first row in the array
409     * String name = rowdata[columnNameList.indexOf("NAME")];
410     *
411     * </code>
412     * @param host
413     * @param query
414     * @param values
415     * @param page
416     * @param rowsPerPage
417     * @param columnNameList
418     * @return
419     * @throws DBException
420     * @throws DBFinderException
421     */
422     @SuppressWarnings("unchecked")
423     public Iterator<String> executeQuery(DBHost host, String query,
424     int rowsPerPage, int page, int rowsPerPage,
425     final List<String> columnNameList) throws DBException,
426     DBFinderException {
427         Session session = null;
428         // boolean isNewSession = false;
429         try {
430             if (query.trim().toLowerCase().startsWith("select")) {
431                 throw new DBException(
432                     "executeQuery() can only execute SELECT queries. Use executeUpdate() instead.");
433             } else if (query.indexOf(';') != -1) {
434                 throw new DBException(
435                     "executeQuery() does not support semi-colons(;).");
436             }
437             // isNewSession = (threadLocalSession.get() == null ||
438             // threadLocalSession.get().getHost().getHome() == null);
439             session = getSession(host);
440             String optimizedQuery = query;
441             // Need to check for IN sequence calls.
442             if (query.toLowerCase().indexOf("IN INTO") == -1) {
443                 optimizedQuery = DAOUtils.getPaginationSQL(query, page,
444                 rowsPerPage, host.getDriver());
445             }
446             final StringBuffer sb = new StringBuffer();
447             List<String> list = session.executeQuery(optimizedQuery, values,
448             new SequenceResultMapper() {
449                 public Object createAndPopulate(ResultSet rs) {
450                     List<String> list = new ArrayList<String>();
451                     sb.append("<tr>");
452                     for (int i = 0; i < rs.getMetaData().getColumnCount(); i++) {
453                         String value = rs.getString(i);
454                         // handle blob column type
455                         if (value != null) {
456                             Blob metaDataBlob = rs.getMetaData().getBlob(i);
457                             int MAX_BLOB_LENGTH = 10240;
458                             Blob blob = rs.getBlob(i);
459                             if (blob != null) {
460                                 if (blob.length() < MAX_BLOB_LENGTH)
461                                     value = new String(blob.getBytes(1,
462                                         (int) blob.length()));
463                                 else
464                                     value = new String(blob.getBytes(1,
465                                         MAX_BLOB_LENGTH));
466                             }
467                         }
468                         list.add(value);
469                         sb.append("<td>").append(value).append("</td>");
470                     }
471                     sb.append("</tr>");
472                     return list.toArray(new String[0]);
473                 }
474             });
475             @Override
476             public void storeResultSetMetaData(
477                 ResultSetMetaData metadata) throws SQLException {
478                 if (columnNameList != null) {
479                     columnNameList.clear();
480                     for (int i = 1; i <= metadata.getColumnCount(); i++) {
481                         columnNameList.add(metadata
482                             .getColumnName(i));
483                     }
484                 }
485                 sb.append("<thead><tr>" + "<th>" + "</th>" + "</tr>" + "</thead>" + "<tbody>");
486                 for (int i = 1; i <= metadata.getColumnCount(); i++) {
487                     sb.append("<th>" + metadata.getColumnName(i) + "</th>" + "</tr>" + "<tbody>");
488                 }
489             }
490             sb.append("</tbody></table>");
491             String filename = StringHelper.getRandHashCode("DB");
492             String htmlPath = ContextManager.getContext().getGlobalContext()
493                 .getOutputDir(DIRECTORY) + "/HTML/" + filename + ".html";
494             _writeToFile(htmlPath, "HTML", title, query);
495         }
496     }

```

```

497             String filename = StringHelper.getRandHashCode("DB");
498             String htmlPath = ContextManager.getContext().getGlobalContext()
499                 .getOutputDir(DIRECTORY) + "/HTML/" + filename + ".html";
500             _writeToFile(htmlPath, "HTML", title, query);
501         }
502     }
503     if (ContextManager.getGlobalContext() != null
504         && ContextManager.getGlobalContext().getAttribute(
505             Context.OUTPUT_DIRECTORY) != null
506         && this.isLoggedInIn() {
507             String filename = StringHelper.getRandHashCode("DB");
508             String htmlPath = ContextManager.getContext().getGlobalContext()
509                 .getOutputDir(DIRECTORY) + "/HTML/" + filename + ".html";
510             _writeToFile(htmlPath, "HTML", title, query);
511         }
512     }
513     _writeToFile(file,
514     "HTML",
515     "HTML<title>" + title + "</title>" + query);
516 }

```

```

510     String spath = ContextManager.getGlobalContext()
511         .getAttributedContext().OUTPUT_DIRECTORY
512         + "/html/"
513         + filename + ".html";
514     _writer.print(spath);
515     _writer.print(srcPath,
516         "<html><title>" +
517             "Query</title><body>" +
518             "<table width=100% border=1 cellspacing=1 bgcolor=silver>" +
519             "<tr><td>Host</td><td>" +
520             host.getName() +
521             "</td></tr>" +
522             "<tr><td>Query:</td><td>" +
523             query +
524             "</td></tr>" +
525             "<tr><td>Form:</td><td>" +
526             Utils.dumpArray(values) +
527             "</td></tr>" +
528             "<tr><td>Returned:</td><td>" +
529             (null == list ? list : list.size() > 0 ? list
530                 .size() : 0) +
531             "</td></tr><table border=1 cellpadding=1>" +
532             "<tr><td>" +
533             Utils.dumpArray(list) +
534             "</td></tr></table>" +
535             "</body></html>");
536 
537 Logging.logInfo("Host:" +
538     host.getName() +
539     " Query=" +
540     query +
541     " values=" +
542     Utils.dumpArray(values) +
543     ", " +
544     (null != list && list.size() > 0 ? " a href=" +
545         ContextManager.getGlobalContext().getSuiteO()
546         .getTestNContext().getSuiteO()
547         .getName() + ".html" + filename +
548         "#row" + targetIndex + list.size() +
549         " row" + i : "# row" + i) + " returned.");
550 }
551 
552 if (list != null)
553     return list.iterator();
554 
555 throw new DBFinderException("No record(s) found.");
556 } catch (SequiduleDBFinderException e) {
557     Logging.logError("DBFinderException: " + e.getMessage());
558     Logging.logError("hostName=" + host.getName() + ", Query=" + query
559     + ", values=" + Utils.dumpArray(values)
560     + ", " + list.size() + " rows returned.", false);
561     throw new DBFinderException(e);
562 } catch (Exception e) {
563     Logging.logInfo("Host=" + host.getName() + ", Query=" + query
564     + ", values=" + Utils.dumpArray(values) + "!", true);
565     throw new DBException(e);
566 }
567 
568 // there is no need to close the closure here -- michael.wu
569 /**
570 * Finially if (isnewSession) { logger.info("a" + ContextManager.getThreadContext().getMethodInfo() + " - Cleaning up DB Session [" + session); try { close(host); } catch { (Exception e) { // KEEP ME } } }
571 */
572 }
573 /**
574 * Execute a query and returns an Iterator with results, only first 50 rows
575 * returned. It can also populate an array list with column names. This is
576 * convenient to be used by test author. Example: <code>
577 * <pre>
578 * List<String> columnNamesList = new ArrayList<String>();
579 * Iterator<String[]> resultsIterator = executeQuery(DBHosts.StagingDB.UserHostReadPhysical,
580 *         "SELECT * FROM EMP_TABLE WHERE ID=? AND NAME=?",
581 *         new Object[]{12345, "John"}, columnNamesList);
582 * 
583 * String[] rowData = resultsIterator.next();
584 * String id = rowData[columnNamesList.indexOf("ID")];//Retrieve the id & name for the first row in the array
585 * String name = rowData[columnNamesList.indexOf("NAME")];
586 * </pre>
587 * <code>
588 *     @param host
589 *     @param query
590 *     @param values
591 *     @param columnNamesList
592 *     @return
593 *     @throws DBFinderException
594 *     @throws DBException
595 * />
596 public Iterator<String[]> executeQuery(DBHost host, String query,
597     Object[] values, List<String> columnNamesList) throws DBException,
598     DBFinderException {
599     return executeQuery(host, query, values, 1, 50, columnNamesList);
600 }
601 
602 /**
603 * Execute a query and return true if it returns 0 row. This method is used
604 * to check whether existing row before doing insert/update/delete. Example:
605 * <code>
606 * <pre>
607 * boolean zeroResult = executeQuery(DBHosts.StagingDB.UserHostReadPhysical,
608 *         "SELECT * FROM EMP_TABLE WHERE ID=123");
609 * </pre>
610 * <code>
611 *     @param host
612 *     @param query
613 *     @return
614 *     @throws DBException
615 * />
616 public boolean checkZeroQueryResult(DBHost host, String query)
617     throws DBException {
618     return checkZeroQueryResult(host, query, null);
619 }
620 
621 /**
622 * Execute a query and return true if it returns 0 row. This method is used
623 * to check whether existing row before doing insert/update/delete. Example:
624 * <code>
625 * <pre>
626 * boolean zeroResult = executeQuery(DBHosts.StagingDB.UserHostReadPhysical,
627 *         "SELECT * FROM EMP_TABLE WHERE ID=? AND NAME=?",
628 *         new Object[]{12345, "John"});
629 * </pre>
630 * <code>
631 *     @param host
632 *     @param query
633 *     @param values
634 *     @return
635 *     @throws DBException
636 * />

```

```

577 }
578 /**
579 * Execute a query and returns an Iterator with results, only first 50 rows
580 * returned. It can also populate an array list with column names. This is
581 * convenient to be used by test author. Example: <code>
582 * <pre>
583 * List<String> columnNamesList = new ArrayList<String>();
584 * Iterator<String[]> resultsIterator = executeQuery(DBHosts.StagingDB.UserHostReadPhysical,
585 *         "SELECT * FROM EMP_TABLE WHERE ID=? AND NAME=?",
586 *         new Object[]{12345, "John"}, columnNamesList);
587 * 
588 * String[] rowData = resultsIterator.next();
589 * String id = rowData[columnNamesList.indexOf("ID")];//Retrieve the id & name for the first row in the array
590 * String name = rowData[columnNamesList.indexOf("NAME")];
591 * </pre>
592 * <code>
593 *     @param host
594 *     @param query
595 *     @param values
596 *     @param columnNamesList
597 *     @return
598 *     @throws DBFinderException
599 *     @throws DBException
600 * />
601 public Iterator<String[]> executeQuery(DBHost host, String query,
602     Object[] values, List<String> columnNamesList) throws DBException,
603     DBFinderException {
604     return executeQuery(host, query, values, 1, 50, columnNamesList);
605 }
606 
607 /**
608 * Execute a query and return true if it returns 0 row. This method is used
609 * to check whether existing row before doing insert/update/delete. Example:
610 * <code>
611 * <pre>
612 * boolean zeroResult = executeQuery(DBHosts.StagingDB.UserHostReadPhysical,
613 *         "SELECT * FROM EMP_TABLE WHERE ID=123");
614 * </pre>
615 * <code>
616 *     @param host
617 *     @param query
618 *     @return
619 *     @throws DBException
620 * />
621 public boolean checkZeroQueryResult(DBHost host, String query)
622     throws DBException {
623     return checkZeroQueryResult(host, query, null);
624 }
625 
626 /**
627 * Execute a query and return true if it returns 0 row. This method is used
628 * to check whether existing row before doing insert/update/delete. Example:
629 * <code>
630 * <pre>
631 * boolean zeroResult = executeQuery(DBHosts.StagingDB.UserHostReadPhysical,
632 *         "SELECT * FROM EMP_TABLE WHERE ID=? AND NAME=?",
633 *         new Object[]{12345, "John"});
634 * </pre>
635 * <code>
636 *     @param host
637 *     @param query
638 *     @param values
639 *     @return
640 *     @throws DBException
641 * />

```

```

539     * @param values
540     * @return
541     * @throws DBException
542     */
543    public boolean check1ZeroQueryResult(DBHost host, String query,
544        Object[] values) throws DBException {
545        try {
546            executeQuery(host, query, values, 1, 1, null);
547        } catch (DBIndexException ex) {
548            return true;
549        }
550        return false;
551    }
552    /**
553     * Execute on update / delete statement and returns the number of database
554     * table rows effected. Example: <code>
555     * <pre>
556     * int rows = executeUpdate(DBHosts.StagingDB.UserHostWritePhysical,
557     *                         "UPDATE EMP_TABLE SET NAME='Abhijit' WHERE ID=50");
558     * </pre>
559     * </code>
560     *
561     * @param host
562     * @param query
563     * @return
564    */
565    public int executeUpdate(DBHost host, String query) throws DBException {
566        return executeUpdate(host, query, null);
567    }
568    /**
569     * Execute on update / delete statement and returns the number of database
570     * table rows effected. Provides option to commit the DML. Example: <code>
571     * <pre>
572     * int rows = executeUpdate(DBHosts.StagingDB.UserHostsWritePhysical,
573     *                         "UPDATE EMP_TABLE SET NAME='Abhijit' WHERE ID=50", true);
574     * </pre>
575     * </code>
576     *
577     * @param host
578     * @param query
579     * @param autocommit
580     * @return
581    */
582    public int executeUpdate(DBHost host, String query, boolean autocommit)
583        throws DBException {
584        int rows = executeUpdate(host, query, null);
585        if (autocommit)
586            commit(host);
587        return rows;
588    }
589    /**
590     * Execute on update / delete statement and returns the number of database
591     * table rows effected. Example: <code>
592     * <pre>
593     * int rows = executeUpdate(DBHosts.StagingDB.UserHostsWritePhysical,
594     *                         "UPDATE EMP_TABLE SET NAME='John' WHERE ID=7",
595     *                         new Object[]{"John", 12345});
596     * </pre>
597     * </code>
598     *
599     * @param host
600     * @param query
601     * @param values
602    */

```

```

700     * @param values
701     * @return
702     * @throws Exception
703    }
704    public int executeUpdate(DBHost host, String query, Object[] values)
705        throws DBException {
706        try {
707            String copyQuery = query.trim().toUpperCase();
708            if (!copyQuery.startsWith("SELECT"))
709                //& commented if copyQuery.indexOf("UPDATE") == -1 {
710                throw new DBException("executeUpdate() can not execute SELECT queries. Use executeQuery() instead.");
711            else if (query.indexOf(';') != -1) {
712                throw new DBException("executeUpdate() does not support semi-colons(). Please issue individual DMLs separately.");
713            }
714            Session session = getSession(host);
715            int result = session.executeUpdate(query, values);
716            if (this.isLogEnabled())
717                Logging.logBCall("Host=" + host.getName() + ", Query=" + query
718                    + ", values=" + Utils.dumpArray(values) + ")", true);
719            else
720                result += row(result);
721            return result;
722        } catch (SQLException e) {
723            Logging.logBCall("Host=" + host.getName() + ", Query=" + query
724                    + ", values=" + Utils.dumpArray(values) + ")", true);
725            throw new DBException(e);
726        }
727    }
728    private boolean isLogEnabled() {
729        return ContextManager.getThreadContext().isDBLogEnabled();
730    }
731    public void loadConfigProperties(Map prop, DBHost hostInfo) {
732        String jdbcContext = hostInfo.getJdbcContext();
733        prop.put(jdbcContext + ".jdbc.driver", hostInfo.getDriver());
734        if (hostInfo.getJdbc() != null)
735            prop.put(jdbcContext + ".jdbc.url",
736                "jdbc:mysql://" + hostInfo.getHostAddress() + ":" +
737                hostInfo.getPort() + "#" + hostInfo.getId());
738        else
739            prop.put(jdbcContext + ".jdbc.url",
740                hostInfo.getJdbcContext() + ".jdbc.url");
741        prop.put(jdbcContext + ".jdbc.user", hostInfo.getUser());
742        prop.put(jdbcContext + ".jdbc.password", hostInfo.getPassword());
743        prop.put(jdbcContext + ".jdbc.autocommit", "false");
744        prop.put(jdbcContext + ".pool.maxsize", "50");
745    }
746    /**
747     * Rolls back to the DBHost for the current thread.
748     * @throws DBException
749    */
750    public void rollback() throws DBException {
751        if (threadLocalDBHostStack.get() == null ||
752            !threadLocalDBHostStack.get().isEmpty()) throw new
753            DBException("you haven't executed a query on any DBHosts yet.");
754    }
755    if (threadLocalDBHostStack.get() != null
756        && !threadLocalDBHostStack.get().isEmpty())
757        rollback(threadLocalDBHostStack.get().peek());

```



The screenshot shows an IDE interface with multiple tabs open, displaying Java code for various MongoDB driver classes. The tabs include ResponseFactory.class, RestClient.class, RestClientReportLogger.class, RESTResponse.class, DBDriver.class, and MongoDBDriver.class. The code is annotated with numerous Javadoc-style comments, many of which are marked as deprecated. The code handles session management, query execution, and connection pooling.

```
718     int result = session.executeUpdateQuery(values);
719     if (!this.isLogEnabled())
720         Logging.logInfo("Host=" + host.getHostName() + ", Query=" +
721                         query + ", values=" + Utils.dumpArray(values) +
722                         " ) " + result + " rows updated.", false);
723
724     return result;
725 } catch (SQLException e) {
726     Logging.logError("Host=" + host.getHostName() + ", Query=" + query +
727                     ", values=" + Utils.dumpArray(values) + ")", true);
728     throw new DBException(e);
729 }
730 }
731
732 /**
733  * private boolean isLogEnabled() {
734  *     return ContextManager.getThreadContext().isDBLogEnabled();
735  * }
736
737 public void loadHostConfig(Properties prop, DBHost hostInfo) {
738     String jdbcContext = hostInfo.getHostName();
739     prop.put(jdbcContext + ".jdbc_driver", hostInfo.getDriver());
740     if (hostInfo.getUO() != null)
741         prop.put(jdbcContext + ".jdbc_url",
742                         "jdbc:oracle:thin:@"
743                         + hostInfo.getHostAddress() + ":" +
744                         + hostInfo.getPort() + ":" + hostInfo.getSID());
745     else
746         prop.put(jdbcContext + ".jdbc_url", hostInfo.getUO());
747     prop.put(jdbcContext + ".jdbc_user", hostInfo.getUser());
748     prop.put(jdbcContext + ".jdbc_password", hostInfo.getPassword());
749     prop.put(jdbcContext + ".jdbc_autocommit", "false");
750     prop.put(jdbcContext + ".pool_maxsize", "50");
751
752     /**
753      * Rolls back to the DBHost for the current thread.
754      * @throws DBException
755     */
756     public void rollback() throws DBException {
757         /*
758          * If <threadLocalDBHostStack>.getO == null ||
759          * <threadLocalDBHostStack>.getO().isEmpty() throw new
760          * DBException("You haven't executed a query on any DBHosts yet.");
761         */
762         if (<threadLocalDBHostStack>.getO == null
763             && <threadLocalDBHostStack>.getO().isEmpty())
764             rollback(<threadLocalDBHostStack>.getO().peek());
765     }
766
767     /**
768      * Rolls back to the DBHost.
769      * @param host
770      * @throws DBException
771     */
772     public void rollback(DBHost host) throws DBException {
773         try {
774             Session session = _getSession(host);
775             session.rollback();
776             close(host);
777         } catch (SQLException e) {
778             throw new DBException(e);
779         }
780     }
781 }
782 }
783 }
```