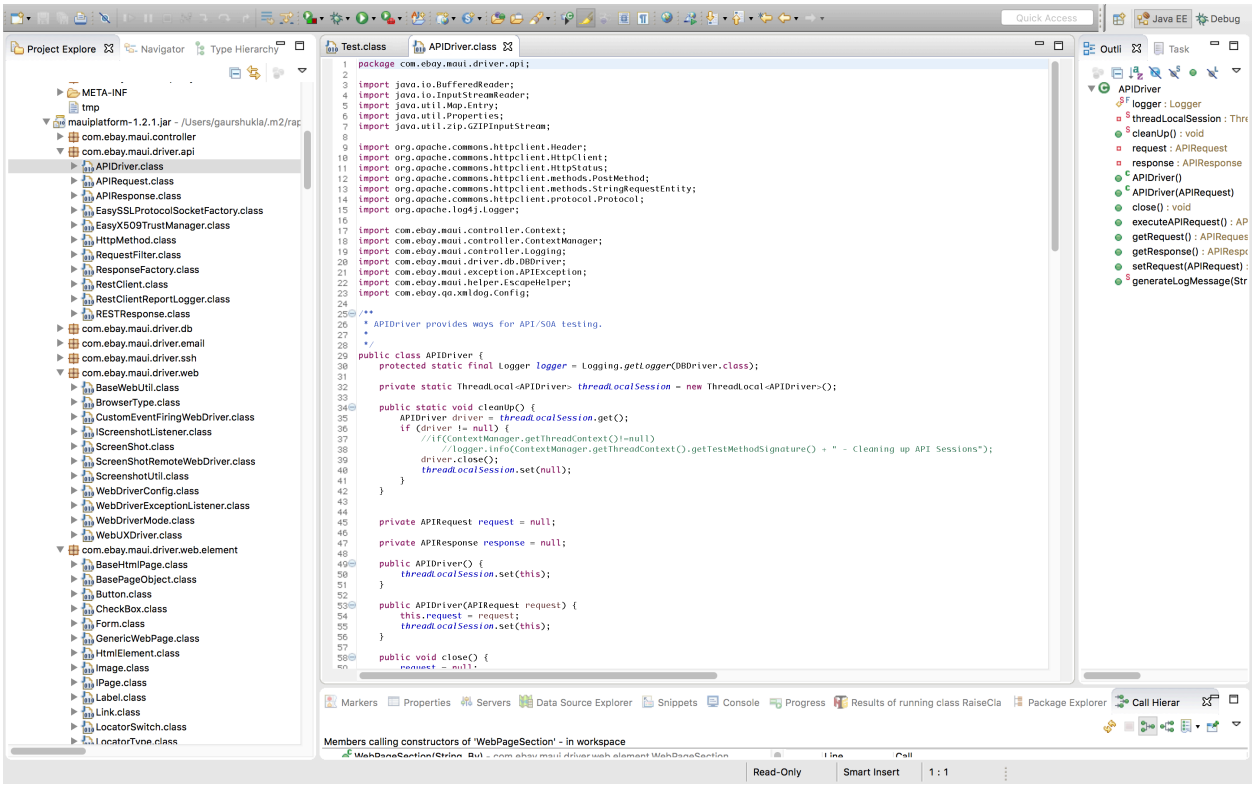
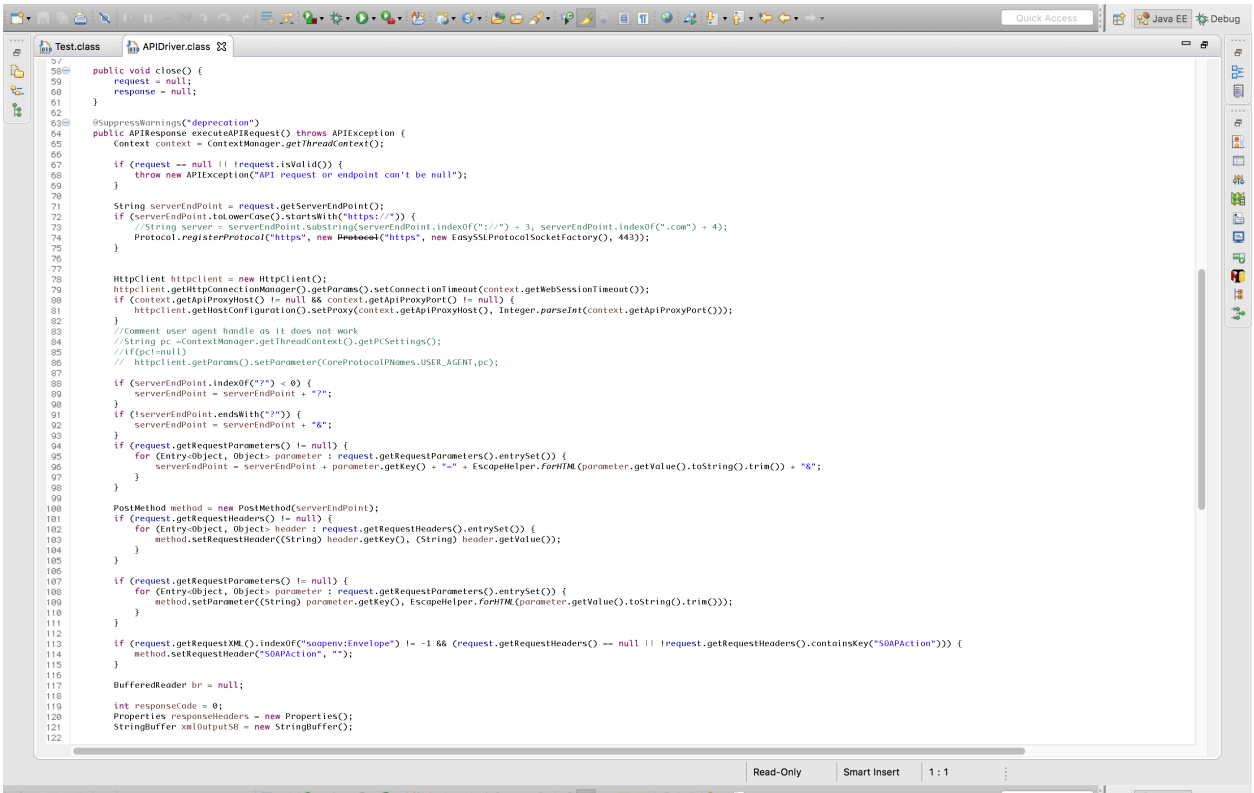
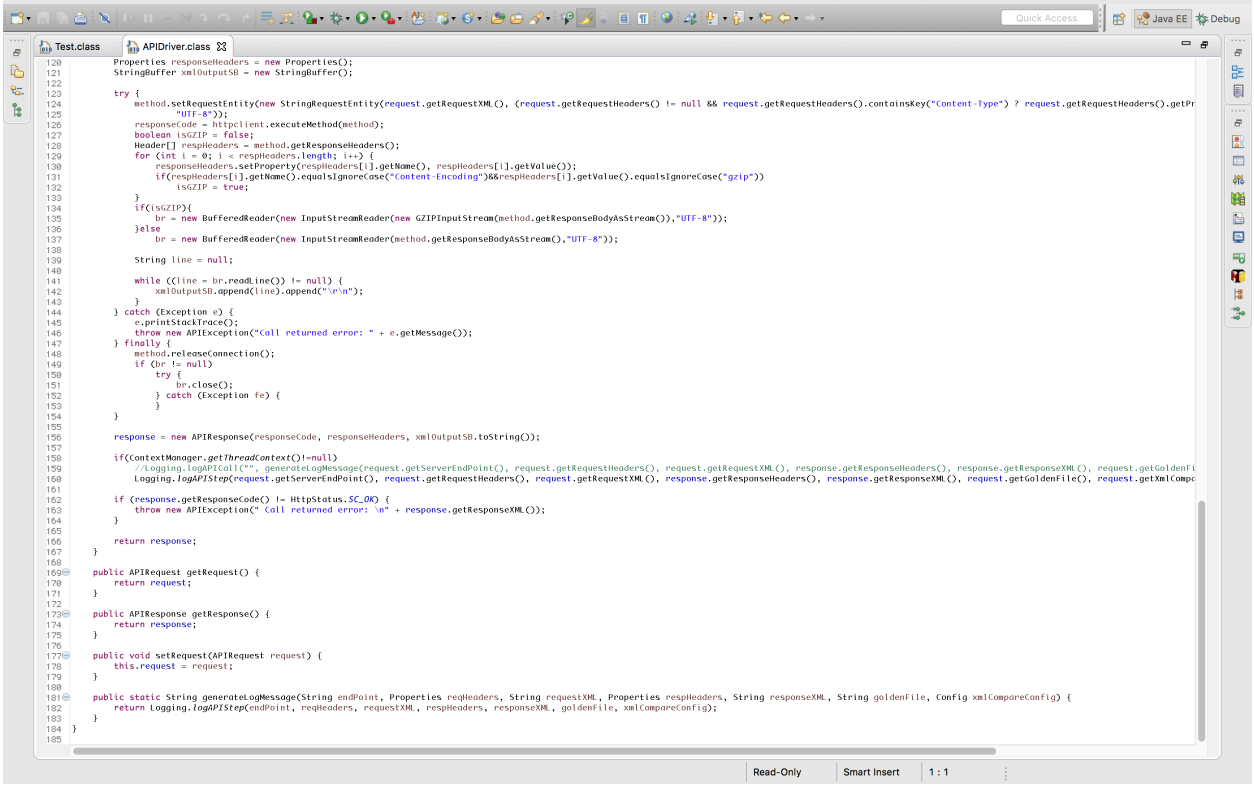
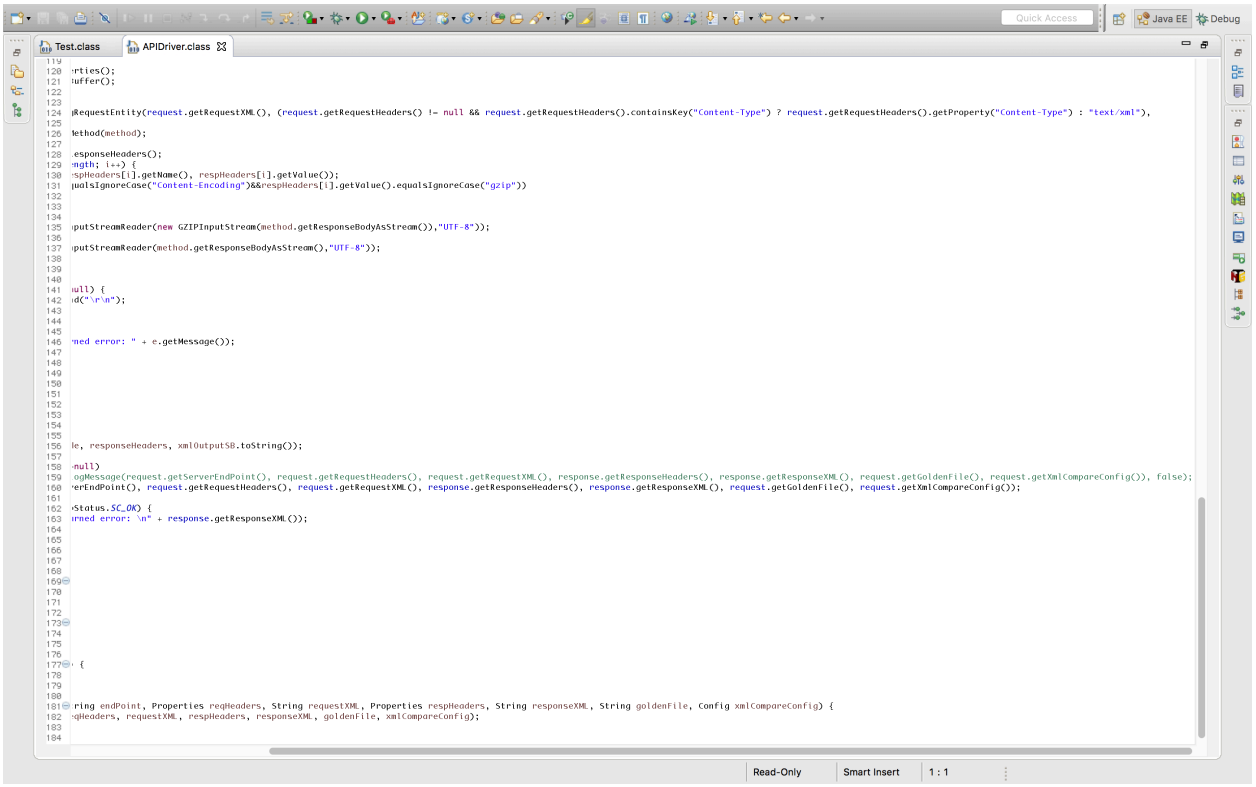

Project Explore
Navigator
Type Hierarchy
META-INF
tmp
mauiplatform-1.2.1.jar - /Users/gaurshukla/.m2/rap
com.ebay.maui.controller
com.ebay.maui.driver.api
APIDriver.class
APIRequest.class
APIResponse.class
EasySSLProtocolSocketFactory.class
EasyX509TrustManager.class
HttpMethod.class
RequestFilter.class
ResponseFactory.class
RestClient.class
RestClientReportLogger.class
RESTResponse.class
com.ebay.maui.driver.db
com.ebay.maui.driver.email
com.ebay.maui.driver.ssh
com.ebay.maui.driver.web
BaseWebUtil.class
BrowserType.class
CustomEventFiringWebDriver.class
IScreenshotListener.class
ScreenShot.class
ScreenShotRemoteWebDriver.class
ScreenshotUtil.class
WebDriverConfig.class
WebDriverExceptionListener.class
WebDriverMode.class
WebUXDriver.class
com.ebay.maui.driver.web.element
BaseHtmlPage.class
BasePageObject.class
Button.class
CheckBox.class
Form.class
GenericWebPage.class
HtmlElement.class
Image.class
IPage.class
Label.class
Link.class
LocatorSwitch.class
Test.class
APIDriver.class
package com.ebay.maui.driver.api;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.util.Map.Entry;
import java.util.Properties;
import java.util.zip.GZIPInputStream;
import org.apache.commons.httpclient.Header;
import org.apache.commons.httpclient.HttpClient;
import org.apache.commons.httpclient.HttpStatus;
import org.apache.commons.httpclient.methods.PostMethod;
import org.apache.commons.httpclient.methods.StringRequestEntity;
import org.apache.commons.httpclient.protocol.Protocol;
import org.apache.log4j.Logger;
import com.ebay.maui.controller.Context;
import com.ebay.maui.controller.ContextManager;
import com.ebay.maui.controller.Logging;
import com.ebay.maui.driver.db.DBDriver;
import com.ebay.maui.exception.APIException;
import com.ebay.maui.helper.EscapeHelper;
import com.ebay.qa.xmldog.Config;
/**
* APIDriver provides ways for API/SOA testing.
*
*/
public class APIDriver {
protected static final Logger logger = Logging.getLogger(DBDriver.class);
private static ThreadLocal<APIDriver> threadLocalSession = new ThreadLocal<APIDriver>();
public static void cleanUp() {
APIDriver driver = threadLocalSession.get();
if (driver != null) {
//if(ContextManager.getThreadContext()!=null)
//logger.info(ContextManager.getThreadContext().getTestMethodSignature() + " - Cleaning up API Sessions");
driver.close();
threadLocalSession.set(null);
}
}
private APIRequest request = null;
private APIResponse response = null;
public APIDriver() {
threadLocalSession.set(this);
}
public APIDriver(APIRequest request) {
this.request = request;
threadLocalSession.set(this);
}
public void close() {
Outli
Task
APIDriver
logger : Logger
threadLocalSession : Thre
cleanUp() : void
request : APIRequest
response : APIResponse
APIDriver()
APIDriver(APIRequest)
close() : void
executeAPIRequest() : AP
getRequest() : APIReques
getResponse() : APIResp
setRequest(APIRequest)
generateLogMessage(Str
Markers
Properties
Servers
Data Source Explorer
Snippets
Console
Progress
Results of running class RaiseCla
Package Explorer
Call Hierar
Members calling constructors of 'WebPageSection' - in workspace
Read-Only
Smart Insert
1 : 1

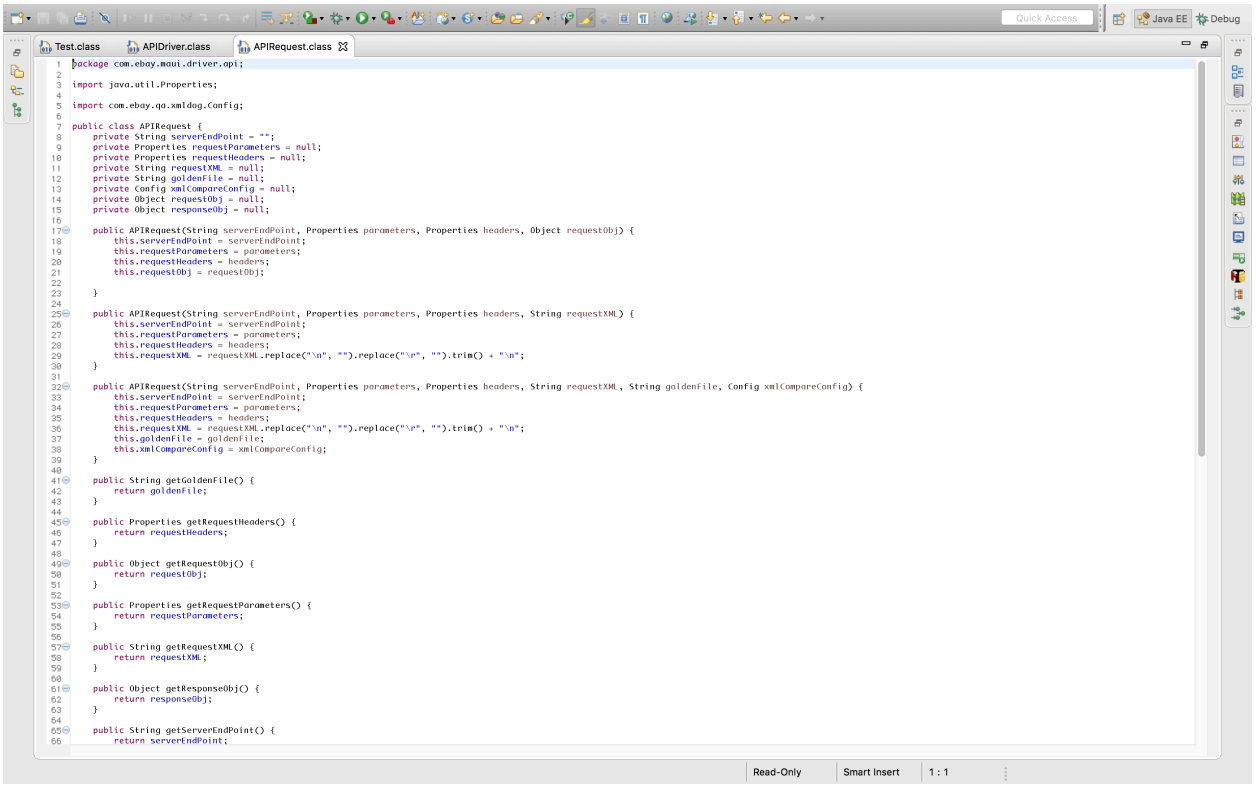


```
Test.class | APIDriver.class

    public void close() {
        request = null;
        response = null;
    }

    @SuppressWarnings("deprecation")
    public APIResponse executeAPIRequest() throws APIException {
        Context context = ContextManager.getThreadContext();

        if (request == null || !request.isValid()) {
            throw new APIException("API request or endpoint can't be null");
        }

        String serverEndPoint = request.getServerEndPoint();
        if (serverEndPoint.toLowerCase().startsWith("https://")) {
            //String server = serverEndPoint.substring(serverEndPoint.indexOf("://") + 3, serverEndPoint.indexOf(".com") + 4);
            Protocol.registerProtocol("https", new ~~Protocol~~("https", new EasySSLProtocolSocketFactory(), 443));
        }


        HttpClient httpclient = new HttpClient();
        httpclient.getHttpConnectionManager().getParams().setConnectionTimeout(context.getWebSessionTimeout());
        if (context.getApiProxyHost() != null && context.getApiProxyPort() != null) {
            httpclient.getHostConfiguration().setProxy(context.getApiProxyHost(), Integer.parseInt(context.getApiProxyPort()));
        }
        //Comment user agent handle as it does not work
        //String pc =ContextManager.getThreadContext().getPCSettings();
        //if(pc!=null)
        //  httpclient.getParams().setParameter(CoreProtocolPNames.USER_AGENT,pc);

        if (serverEndPoint.indexOf("?") < 0) {
            serverEndPoint = serverEndPoint + "?";
        }
        if (!serverEndPoint.endsWith("?")) {
            serverEndPoint = serverEndPoint + "&";
        }
        if (request.getRequestParameters() != null) {
            for (Entry<Object, Object> parameter : request.getRequestParameters().entrySet()) {
                serverEndPoint = serverEndPoint + parameter.getKey() + "=" + EscapeHelper.forHTML(parameter.getValue().toString().trim()) + "&";
            }
        }

        PostMethod method = new PostMethod(serverEndPoint);
        if (request.getRequestHeaders() != null) {
            for (Entry<Object, Object> header : request.getRequestHeaders().entrySet()) {
                method.setRequestHeader((String) header.getKey(), (String) header.getValue());
            }
        }

        if (request.getRequestParameters() != null) {
            for (Entry<Object, Object> parameter : request.getRequestParameters().entrySet()) {
                method.setParameter((String) parameter.getKey(), EscapeHelper.forHTML(parameter.getValue().toString().trim()));
            }
        }

        if (request.getRequestXML().indexOf("soapenv:Envelope") != -1 && (request.getRequestHeaders() == null || !request.getRequestHeaders().containsKey("SOAPAction"))) {
            method.setRequestHeader("SOAPAction", "");
        }

        BufferedReader br = null;

        int responseCode = 0;
        Properties responseHeaders = new Properties();
        StringBuffer xmlOutputSB = new StringBuffer();

Read-Only | Smart Insert | 1 : 1
```

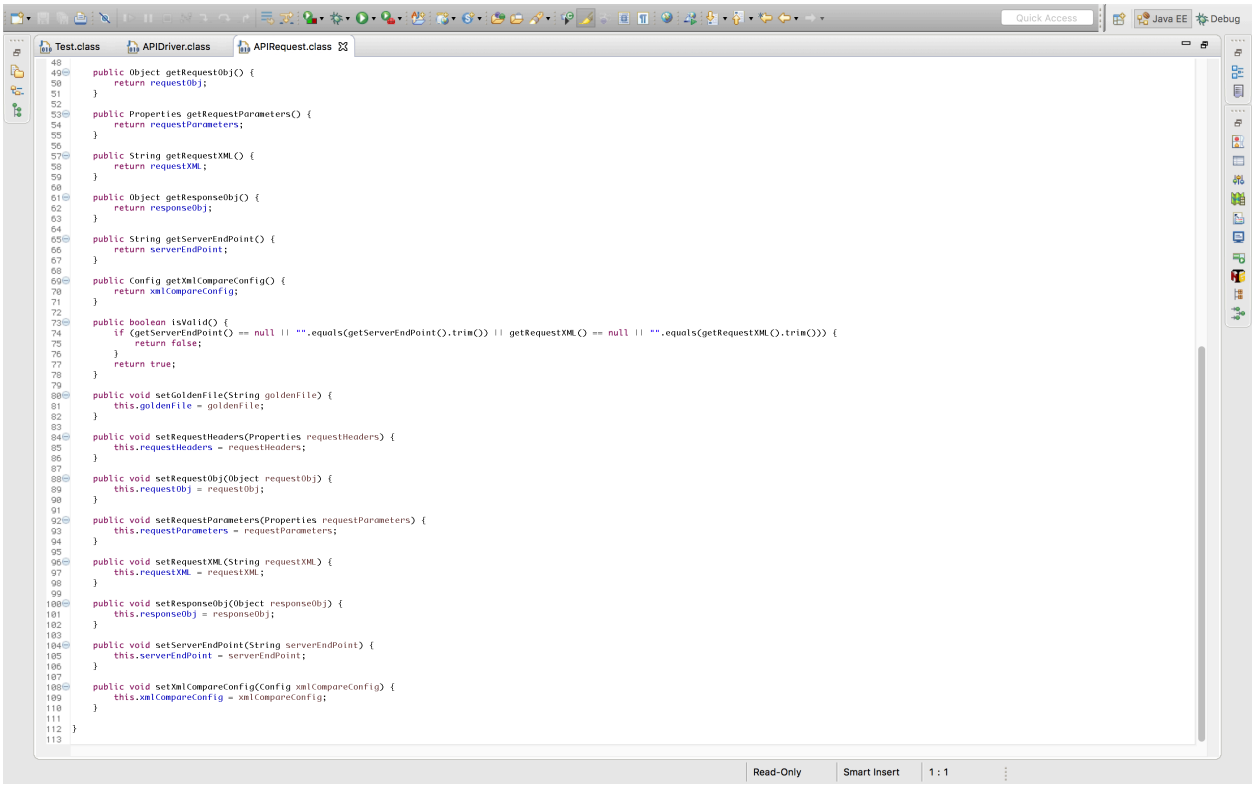


```
Test.class | APIDriver.class

        Properties responseHeaders = new Properties();
        StringBuffer xmlOutputSB = new StringBuffer();

        try {
            method.setRequestEntity(new StringRequestEntity(request.getRequestXML(), (request.getRequestHeaders() != null && request.getRequestHeaders().containsKey("Content-Type") ? request.getRequestHeaders().getPr
                    "UTF-8"));
            responseCode = httpclient.executeMethod(method);
            boolean isGZIP = false;
            Header[] respHeaders = method.getResponseHeaders();
            for (int i = 0; i < respHeaders.length; i++) {
                responseHeaders.setProperty(respHeaders[i].getName(), respHeaders[i].getValue());
                if(respHeaders[i].getName().equalsIgnoreCase("Content-Encoding")&&respHeaders[i].getValue().equalsIgnoreCase("gzip"))
                    isGZIP = true;
            }
            if(isGZIP){
                br = new BufferedReader(new InputStreamReader(new GZIPInputStream(method.getResponseBodyAsStream()),"UTF-8"));
            }else
                br = new BufferedReader(new InputStreamReader(method.getResponseBodyAsStream(),"UTF-8"));

            String line = null;

            while ((line = br.readLine()) != null) {
                xmlOutputSB.append(line).append("\r\n");
            }
        } catch (Exception e) {
            e.printStackTrace();
            throw new APIException("Call returned error: " + e.getMessage());
        } finally {
            method.releaseConnection();
            if (br != null)
                try {
                    br.close();
                } catch (Exception fe) {
                }
        }

        response = new APIResponse(responseCode, responseHeaders, xmlOutputSB.toString());

        if(ContextManager.getThreadContext()!=null)
            //Logging.logAPICall("", generateLogMessage(request.getServerEndPoint(), request.getRequestHeaders(), request.getRequestXML(), response.getResponseHeaders(), response.getResponseXML(), request.getGoldenFi
            Logging.logAPIStep(request.getServerEndPoint(), request.getRequestHeaders(), request.getRequestXML(), response.getResponseHeaders(), response.getResponseXML(), request.getGoldenFile(), request.getXmlCompc

        if (response.getResponseCode() != HttpStatus.SC_OK) {
            throw new APIException(" Call returned error: \n" + response.getResponseXML());
        }

        return response;
    }

    public APIRequest getRequest() {
        return request;
    }

    public APIResponse getResponse() {
        return response;
    }

    public void setRequest(APIRequest request) {
        this.request = request;
    }

    public static String generateLogMessage(String endPoint, Properties reqHeaders, String requestXML, Properties respHeaders, String responseXML, String goldenFile, Config xmlCompareConfig) {
        return Logging.logAPIStep(endPoint, reqHeaders, requestXML, respHeaders, responseXML, goldenFile, xmlCompareConfig);
    }
}

Read-Only | Smart Insert | 1 : 1
```

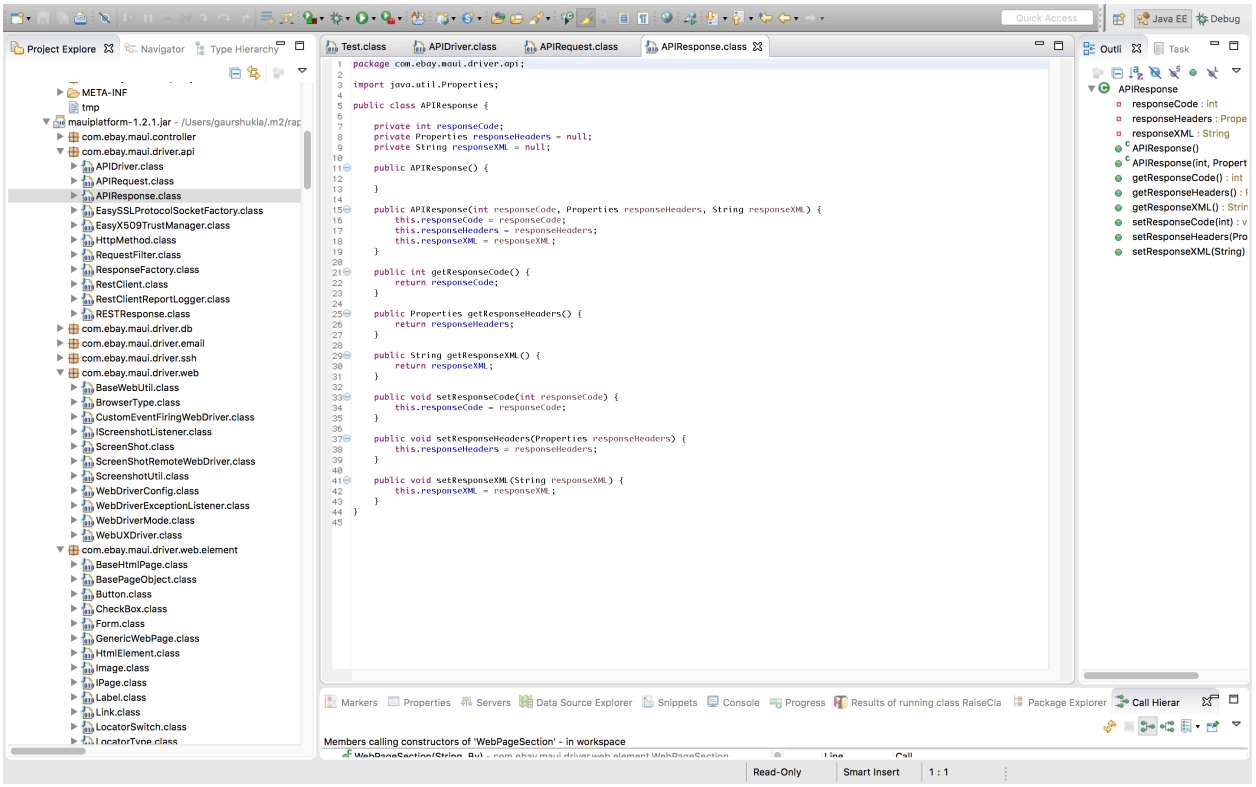

Quick Access
Java EE
Debug
Test.class
APIDriver.class
Read-Only
Smart Insert
1 : 1

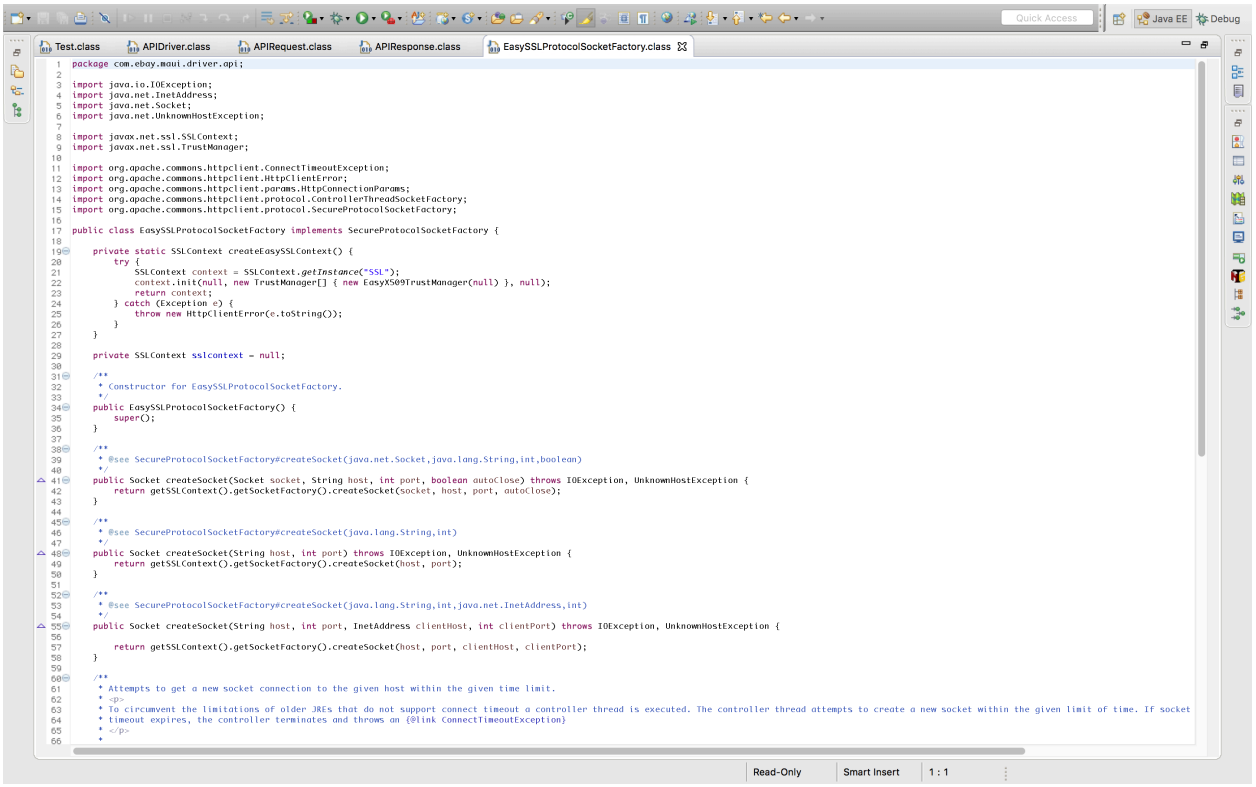


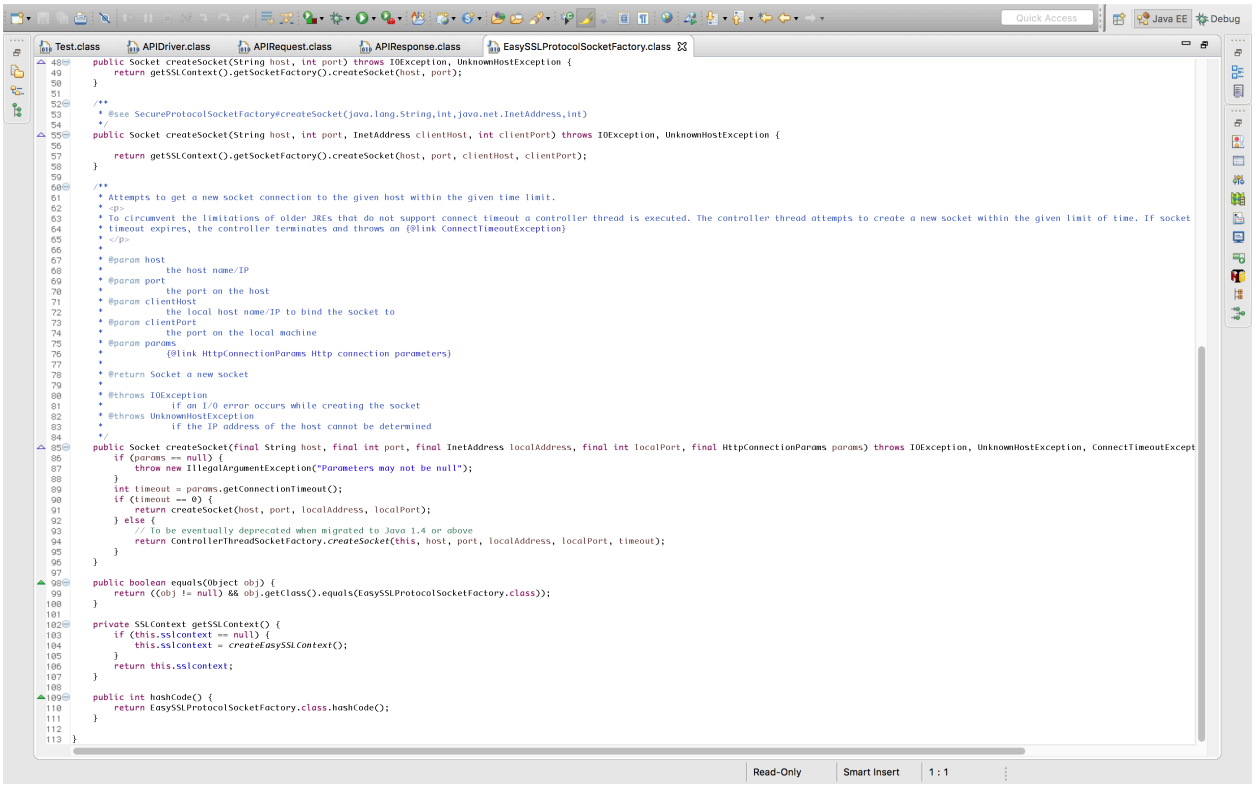
```
package com.ebay.maui.driver.api;

import java.util.Properties;

import com.ebay.qa.xmldog.Config;

public class APIRequest {
    private String serverEndPoint = "";
    private Properties requestParameters = null;
    private Properties requestHeaders = null;
    private String requestXML = null;
    private String goldenFile = null;
    private Config xmlCompareConfig = null;
    private Object requestObj = null;
    private Object responseObj = null;

    public APIRequest(String serverEndPoint, Properties parameters, Properties headers, Object requestObj) {
        this.serverEndPoint = serverEndPoint;
        this.requestParameters = parameters;
        this.requestHeaders = headers;
        this.requestObj = requestObj;

    }

    public APIRequest(String serverEndPoint, Properties parameters, Properties headers, String requestXML) {
        this.serverEndPoint = serverEndPoint;
        this.requestParameters = parameters;
        this.requestHeaders = headers;
        this.requestXML = requestXML.replace("\n", "").replace("\r", "").trim() + "\n";
    }

    public APIRequest(String serverEndPoint, Properties parameters, Properties headers, String requestXML, String goldenFile, Config xmlCompareConfig) {
        this.serverEndPoint = serverEndPoint;
        this.requestParameters = parameters;
        this.requestHeaders = headers;
        this.requestXML = requestXML.replace("\n", "").replace("\r", "").trim() + "\n";
        this.goldenFile = goldenFile;
        this.xmlCompareConfig = xmlCompareConfig;
    }

    public String getGoldenFile() {
        return goldenFile;
    }

    public Properties getRequestHeaders() {
        return requestHeaders;
    }

    public Object getRequestObj() {
        return requestObj;
    }

    public Properties getRequestParameters() {
        return requestParameters;
    }

    public String getRequestXML() {
        return requestXML;
    }

    public Object getResponseObj() {
        return responseObj;
    }

    public String getServerEndPoint() {
        return serverEndPoint;
```



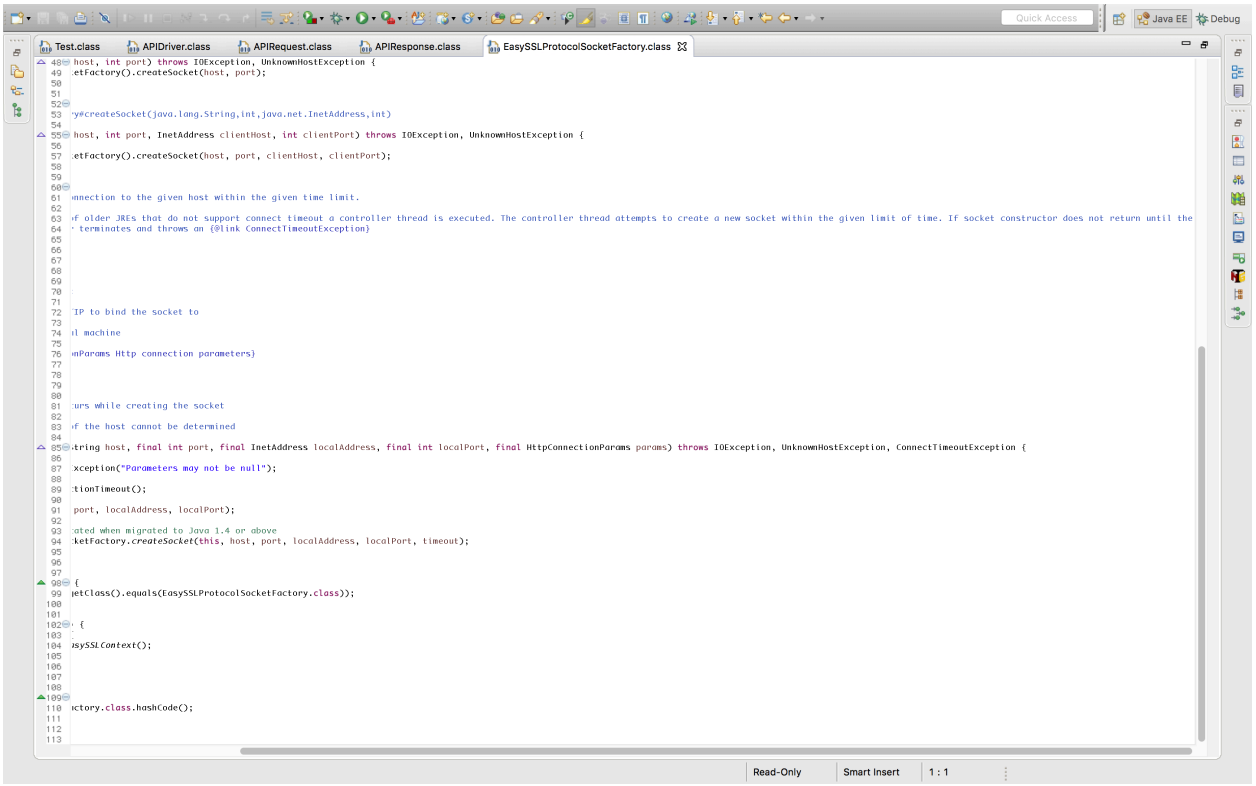
```
    public Object getRequestObj() {
        return requestObj;
    }

    public Properties getRequestParameters() {
        return requestParameters;
    }

    public String getRequestXML() {
        return requestXML;
    }

    public Object getResponseObj() {
        return responseObj;
    }

    public String getServerEndPoint() {
        return serverEndPoint;
    }

    public Config getXmlCompareConfig() {
        return xmlCompareConfig;
    }

    public boolean isValid() {
        if (getServerEndPoint() == null || "".equals(getServerEndPoint().trim()) || getRequestXML() == null || "".equals(getRequestXML().trim())) {
            return false;
        }
        return true;
    }

    public void setGoldenFile(String goldenFile) {
        this.goldenFile = goldenFile;
    }

    public void setRequestHeaders(Properties requestHeaders) {
        this.requestHeaders = requestHeaders;
    }

    public void setRequestObj(Object requestObj) {
        this.requestObj = requestObj;
    }

    public void setRequestParameters(Properties requestParameters) {
        this.requestParameters = requestParameters;
    }

    public void setRequestXML(String requestXML) {
        this.requestXML = requestXML;
    }

    public void setResponseObj(Object responseObj) {
        this.responseObj = responseObj;
    }

    public void setServerEndPoint(String serverEndPoint) {
        this.serverEndPoint = serverEndPoint;
    }

    public void setXmlCompareConfig(Config xmlCompareConfig) {
        this.xmlCompareConfig = xmlCompareConfig;
    }
}
```

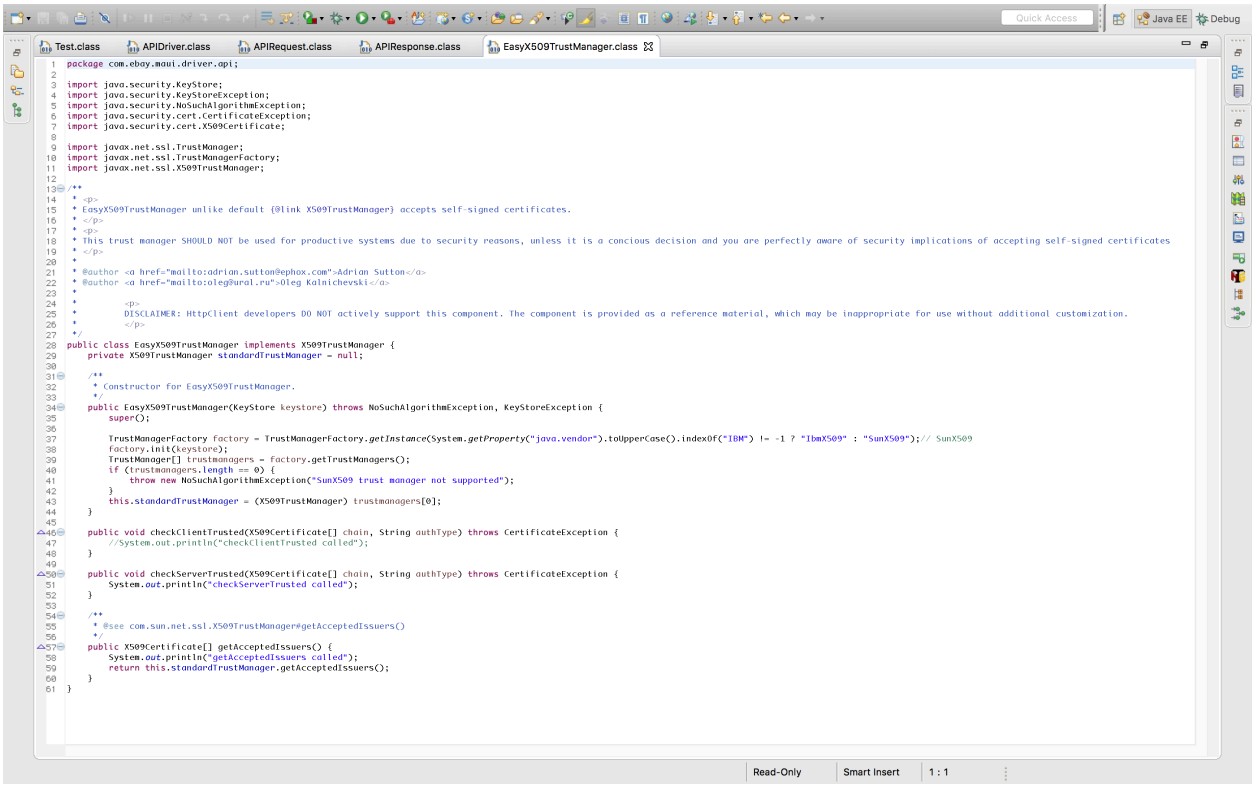

Project Explore
Navigator
Type Hierarchy
META-INF
tmp
mauiplatform-1.2.1.jar - /Users/gaurshukla/.m2/rap
com.ebay.maui.controller
com.ebay.maui.driver.api
APIDriver.class
APIRequest.class
APIResponse.class
EasySSLProtocolSocketFactory.class
EasyX509TrustManager.class
HttpMethod.class
RequestFilter.class
ResponseFactory.class
RestClient.class
RestClientReportLogger.class
RESTResponse.class
com.ebay.maui.driver.db
com.ebay.maui.driver.email
com.ebay.maui.driver.ssh
com.ebay.maui.driver.web
BaseWebUtil.class
BrowserType.class
CustomEventFiringWebDriver.class
IScreenshotListener.class
ScreenShot.class
ScreenShotRemoteWebDriver.class
ScreenshotUtil.class
WebDriverConfig.class
WebDriverExceptionListener.class
WebDriverMode.class
WebUXDriver.class
com.ebay.maui.driver.web.element
BaseHtmlPage.class
BasePageObject.class
Button.class
CheckBox.class
Form.class
GenericWebPage.class
HtmlElement.class
Image.class
IPage.class
Label.class
Link.class
LocatorSwitch.class
Test.class
APIDriver.class
APIRequest.class
APIResponse.class
package com.ebay.maui.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;
}
}
Outl
Task
APIResponse
responseCode : int
responseHeaders : Prope
responseXML : String
APIResponse()
APIResponse(int, Propert
getResponseCode() : int
getResponseHeaders() : I
getResponseXML() : Strin
setResponseCode(int) : v
setResponseHeaders(Pro
setResponseXML(String)
Markers
Properties
Servers
Data Source Explorer
Snippets
Console
Progress
Results of running class RaiseCla
Package Explorer
Call Hierar
Members calling constructors of 'WebPageSection' - in workspace
Read-Only
Smart Insert
1 : 1


Test.class
APIDriver.class
APIRequest.class
APIResponse.class
EasySSLProtocolSocketFactory.class
package com.ebay.maui.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);
return context;
} catch (Exception e) {
throw new HttpClientError(e.toString());
}
}
private SSLContext sslcontext = null;
/**
* Constructor for EasySSLProtocolSocketFactory.
*/
public EasySSLProtocolSocketFactory() {
super();
}
/**
* @see SecureProtocolSocketFactory#createSocket(java.net.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}
*
*
Read-Only
Smart Insert
1 : 1

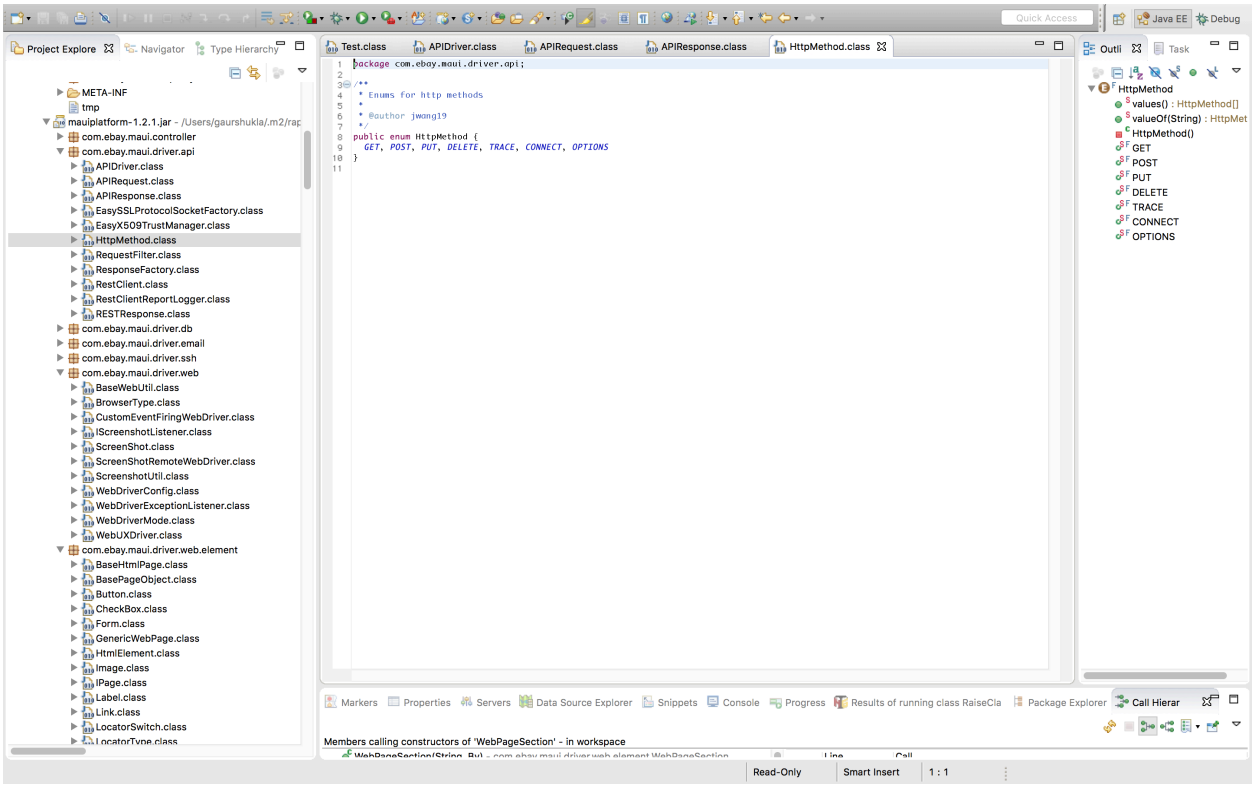



```
    public Socket createSocket(String host, int port) throws IOException, UnknownHostException {
        return getSSLContext().getSocketFactory().createSocket(host, port);
    }

    /**
     * @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.
     * <p>
     * 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}
     * </p>
     *
     * @param host
     *            the host name/IP
     * @param port
     *            the port on the host
     * @param clientHost
     *            the local host name/IP to bind the socket to
     * @param clientPort
     *            the port on the local machine
     * @param params
     *            {@link HttpConnectionParams Http connection parameters}
     *
     * @return Socket a new socket
     *
     * @throws IOException
     *             if an I/O error occurs while creating the socket
     * @throws UnknownHostException
     *             if the IP address of the host cannot be determined
     */
    public Socket createSocket(final String host, final int port, final InetAddress localAddress, final int localPort, final HttpConnectionParams params) throws IOException, UnknownHostException, ConnectTimeoutExcept
        if (params == null) {
            throw new IllegalArgumentException("Parameters may not be null");
        }
        int timeout = params.getConnectionTimeout();
        if (timeout == 0) {
            return createSocket(host, port, localAddress, localPort);
        } else {
            // To be eventually deprecated when migrated to Java 1.4 or above
            return ControllerThreadSocketFactory.createSocket(this, host, port, localAddress, localPort, timeout);
        }
    }

    public boolean equals(Object obj) {
        return ((obj != null) && obj.getClass().equals(EasySSLProtocolSocketFactory.class));
    }

    private SSLContext getSSLContext() {
        if (this.sslcontext == null) {
            this.sslcontext = createEasySSLContext();
        }
        return this.sslcontext;
    }

    public int hashCode() {
        return EasySSLProtocolSocketFactory.class.hashCode();
    }
}
```

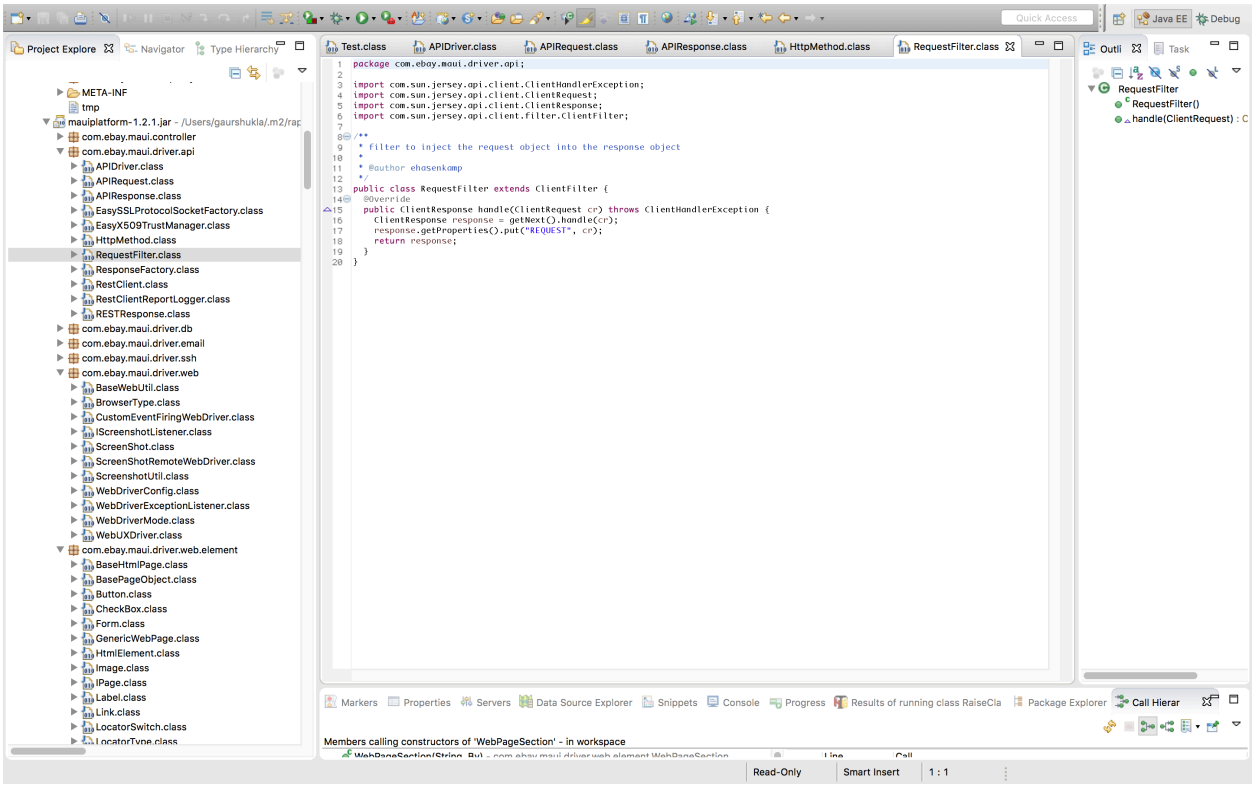


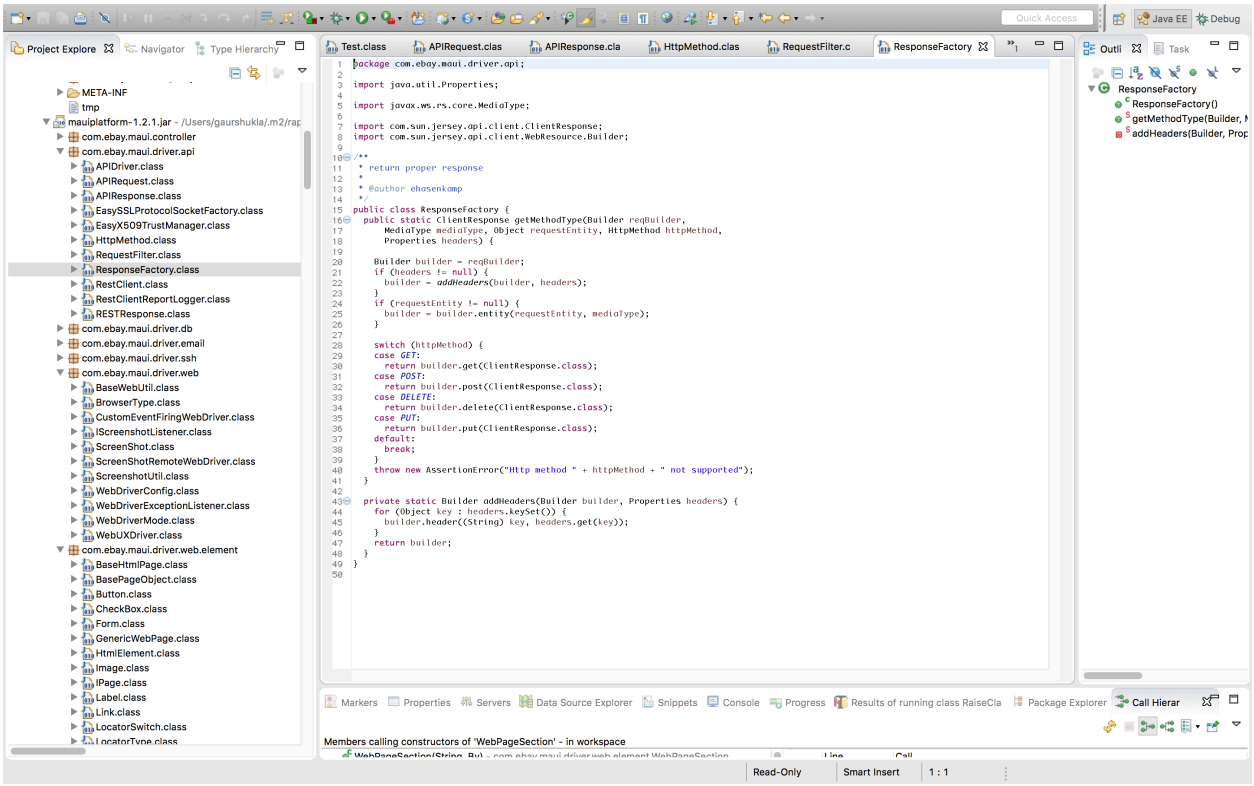


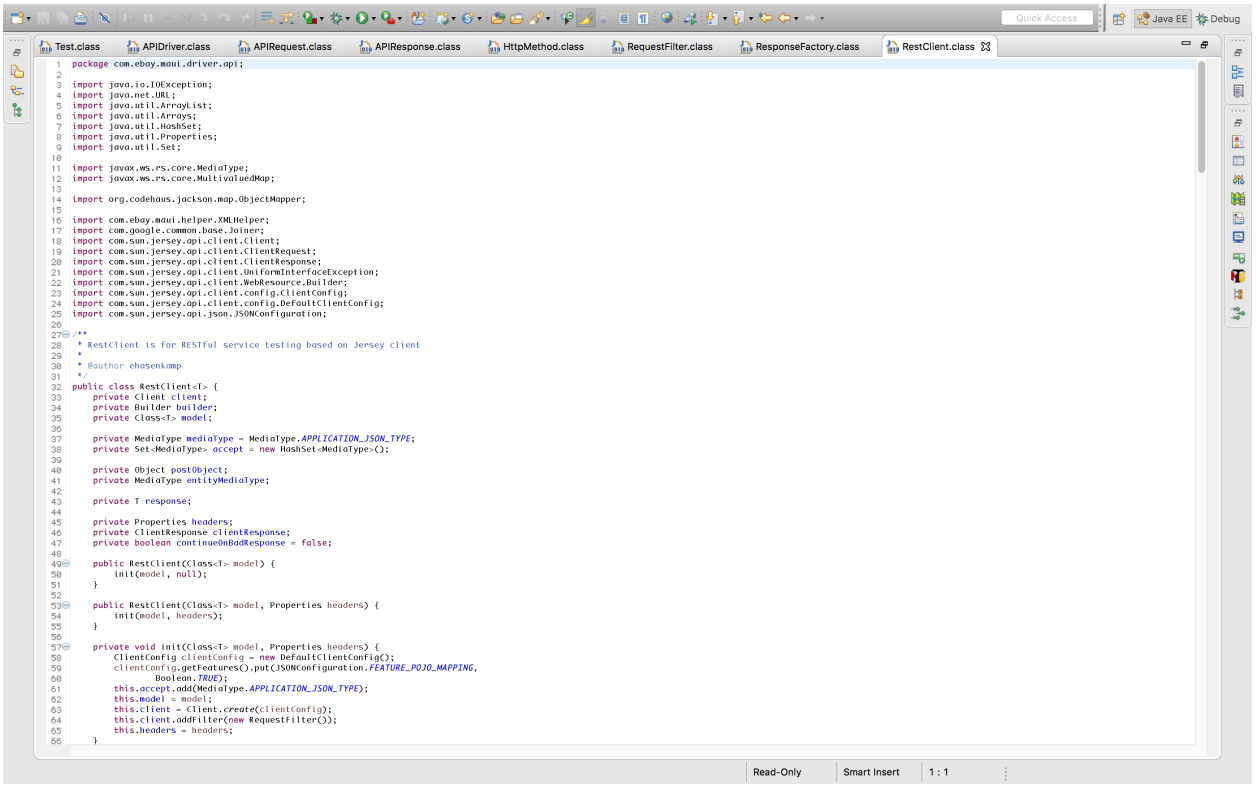

```java
package com.ebay.maui.driver.api;

import java.security.KeyStore;
import java.security.KeyStoreException;
import java.security.NoSuchAlgorithmException;
import java.security.cert.CertificateException;
import java.security.cert.X509Certificate;

import javax.net.ssl.TrustManager;
import javax.net.ssl.TrustManagerFactory;
import javax.net.ssl.X509TrustManager;

/**
 * <p>
 * EasyX509TrustManager unlike default {@link X509TrustManager} accepts self-signed certificates.
 * </p>
 * <p>
 * This trust manager SHOULD NOT be used for productive systems due to security reasons, unless it is a concious decision and you are perfectly aware of security implications of accepting self-signed certificates
 * </p>
 *
 * @author <a href="mailto:adrian.sutton@ephox.com">Adrian Sutton</a>
 * @author <a href="mailto:oleg@ural.ru">Oleg Kalnichevski</a>
 *
 *         <p>
 *         DISCLAIMER: HttpClient developers DO NOT actively support this component. The component is provided as a reference material, which may be inappropriate for use without additional customization.
 *         </p>
 */
public class EasyX509TrustManager implements X509TrustManager {
    private X509TrustManager standardTrustManager = null;

    /**
     * Constructor for EasyX509TrustManager.
     */
    public EasyX509TrustManager(KeyStore keystore) throws NoSuchAlgorithmException, KeyStoreException {
        super();

        TrustManagerFactory factory = TrustManagerFactory.getInstance(System.getProperty("java.vendor").toUpperCase().indexOf("IBM") != -1 ? "IbmX509" : "SunX509");// SunX509
        factory.init(keystore);
        TrustManager[] trustmanagers = factory.getTrustManagers();
        if (trustmanagers.length == 0) {
            throw new NoSuchAlgorithmException("SunX509 trust manager not supported");
        }
        this.standardTrustManager = (X509TrustManager) trustmanagers[0];
    }

    public void checkClientTrusted(X509Certificate[] chain, String authType) throws CertificateException {
        //System.out.println("checkClientTrusted called");
    }

    public void checkServerTrusted(X509Certificate[] chain, String authType) throws CertificateException {
        System.out.println("checkServerTrusted called");
    }

    /**
     * @see com.sun.net.ssl.X509TrustManager#getAcceptedIssuers()
     */
    public X509Certificate[] getAcceptedIssuers() {
        System.out.println("getAcceptedIssuers called");
        return this.standardTrustManager.getAcceptedIssuers();
    }
}
```



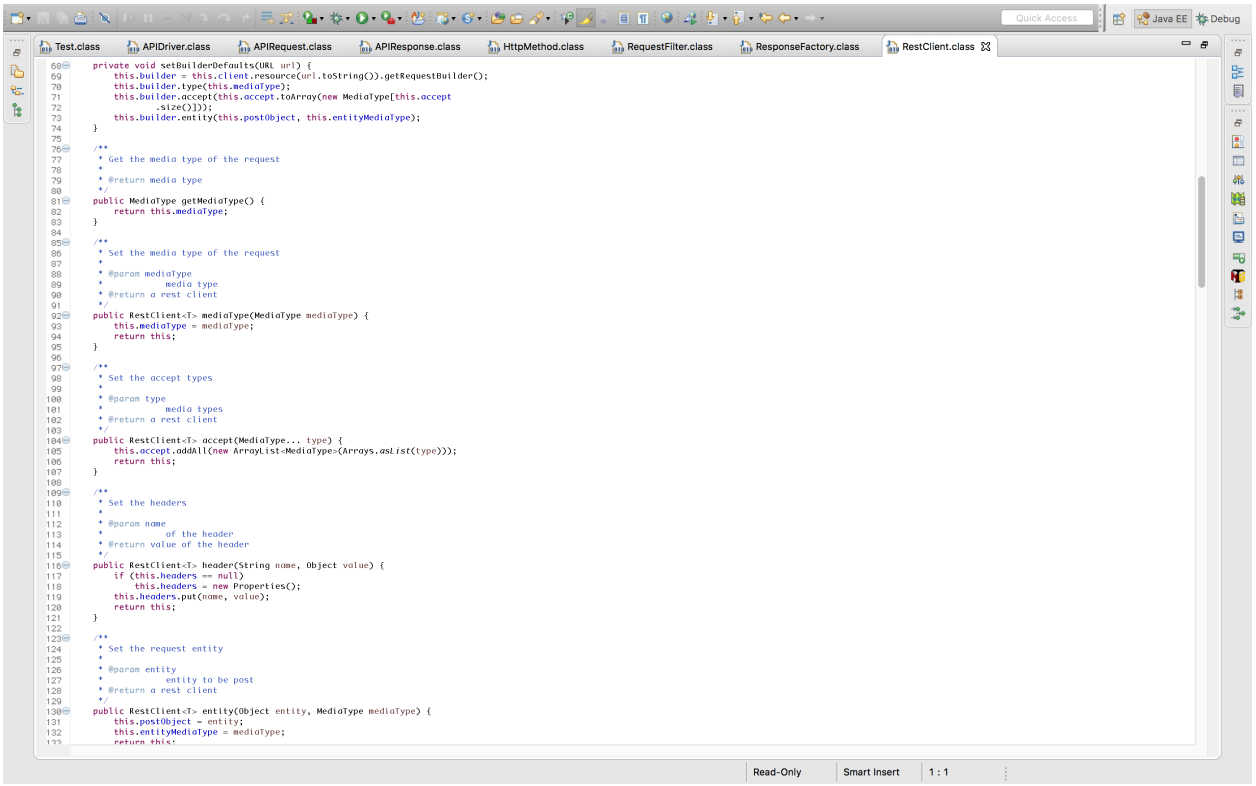

```java
package com.ebay.maui.driver.api;

/**
 * Enums for http methods
 *
 * @author jwang19
 */
public enum HttpMethod {
  GET, POST, PUT, DELETE, TRACE, CONNECT, OPTIONS
}
```


Project Explore
Navigator
Type Hierarchy
Test.class
APIDriver.class
APIRequest.class
APIResponse.class
HttpMethod.class
RequestFilter.class
Outli
Task
RequestFilter
RequestFilter()
handle(ClientRequest) : C
META-INF
tmp
mauiplatform-1.2.1.jar - /Users/gaurshukla/.m2/rap
com.ebay.maui.controller
com.ebay.maui.driver.api
APIDriver.class
APIRequest.class
APIResponse.class
EasySSLProtocolSocketFactory.class
EasyX509TrustManager.class
HttpMethod.class
RequestFilter.class
ResponseFactory.class
RestClient.class
RestClientReportLogger.class
RESTResponse.class
com.ebay.maui.driver.db
com.ebay.maui.driver.email
com.ebay.maui.driver.ssh
com.ebay.maui.driver.web
BaseWebUtil.class
BrowserType.class
CustomEventFiringWebDriver.class
IScreenshotListener.class
ScreenShot.class
ScreenShotRemoteWebDriver.class
ScreenshotUtil.class
WebDriverConfig.class
WebDriverExceptionListener.class
WebDriverMode.class
WebUXDriver.class
com.ebay.maui.driver.web.element
BaseHtmlPage.class
BasePageObject.class
Button.class
CheckBox.class
Form.class
GenericWebPage.class
HtmlElement.class
Image.class
IPage.class
Label.class
Link.class
LocatorSwitch.class
Markers
Properties
Servers
Data Source Explorer
Snippets
Console
Progress
Package Explorer
Call Hierar
Members calling constructors of 'WebPageSection' - in workspace
Read-Only
Smart Insert
1 : 1

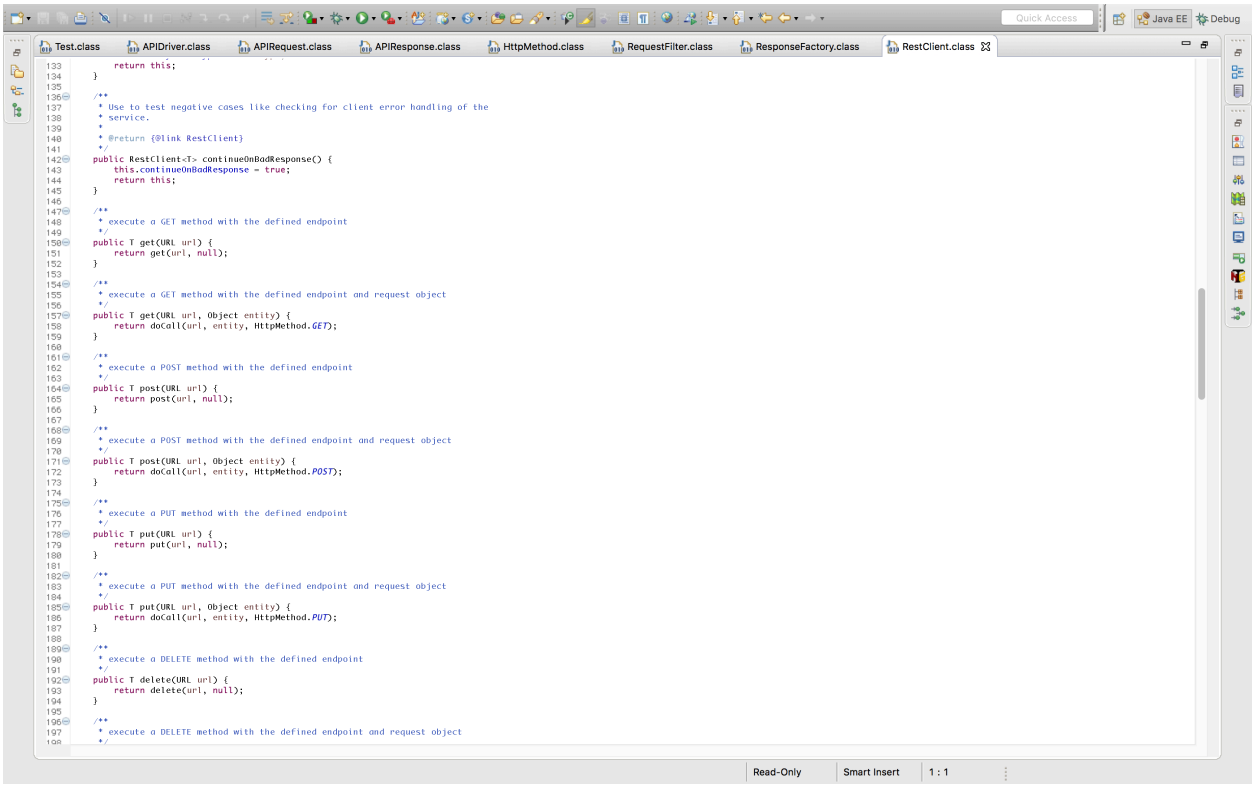

Project Explore
Navigator
Type Hierarchy
Test.class
APIRequest.clas
APIResponse.cla
HttpMethod.clas
RequestFilter.c
ResponseFactory
Outli
Task
ResponseFactory
ResponseFactory()
getMethodType(Builder,
addHeaders(Builder, Prop
META-INF
tmp
mauiplatform-1.2.1.jar - /Users/gaurshukla/.m2/rap
com.ebay.maui.controller
com.ebay.maui.driver.api
APIDriver.class
APIRequest.class
APIResponse.class
EasySSLProtocolSocketFactory.class
EasyX509TrustManager.class
HttpMethod.class
RequestFilter.class
ResponseFactory.class
RestClient.class
RestClientReportLogger.class
RESTResponse.class
com.ebay.maui.driver.db
com.ebay.maui.driver.email
com.ebay.maui.driver.ssh
com.ebay.maui.driver.web
BaseWebUtil.class
BrowserType.class
CustomEventFiringWebDriver.class
IScreenshotListener.class
ScreenShot.class
ScreenShotRemoteWebDriver.class
ScreenshotUtil.class
WebDriverConfig.class
WebDriverExceptionListener.class
WebDriverMode.class
WebUXDriver.class
com.ebay.maui.driver.web.element
BaseHtmlPage.class
BasePageObject.class
Button.class
CheckBox.class
Form.class
GenericWebPage.class
HtmlElement.class
Image.class
IPage.class
Label.class
Link.class
LocatorSwitch.class
Markers
Properties
Servers
Data Source Explorer
Snippets
Console
Progress
Package Explorer
Call Hierar
Members calling constructors of 'WebPageSection' - in workspace
Read-Only
Smart Insert
1 : 1



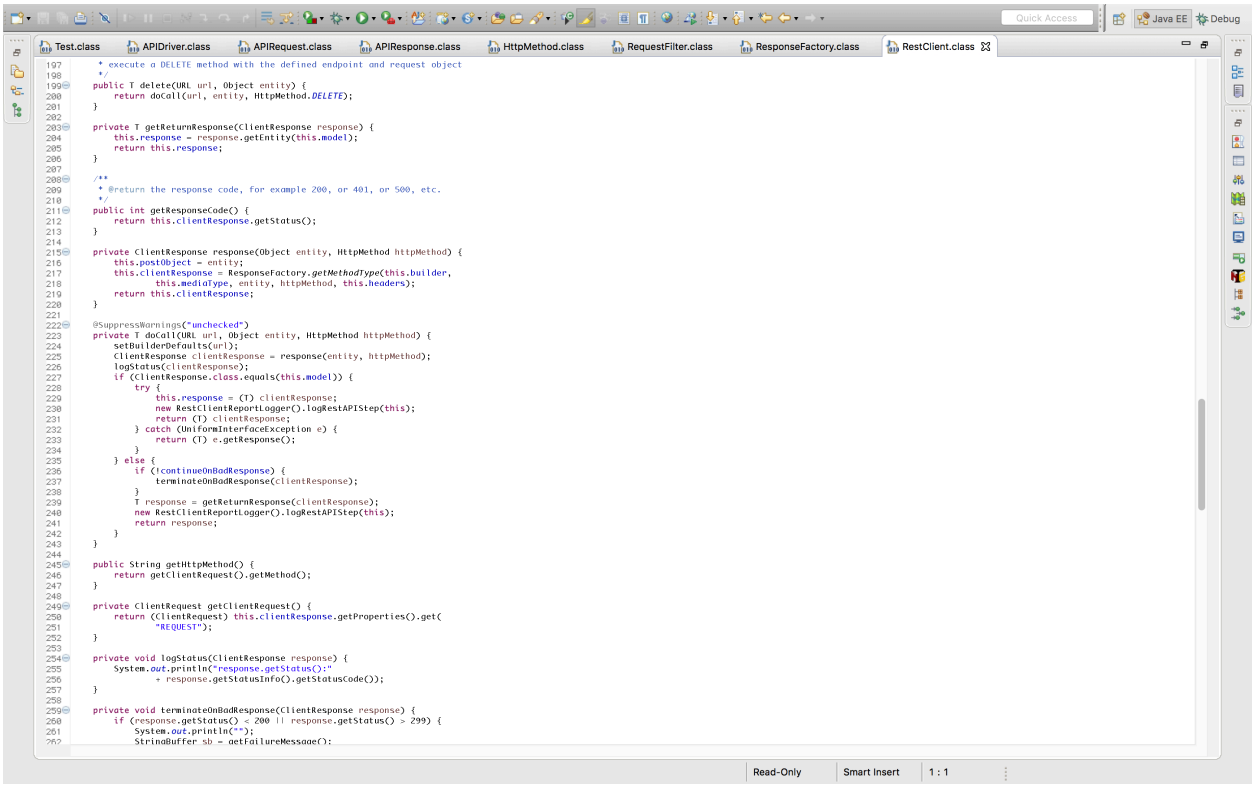

```java
package com.ebay.maui.driver.api;

import java.io.IOException;
import java.net.URL;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashSet;
import java.util.Properties;
import java.util.Set;

import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.MultivaluedMap;

import org.codehaus.jackson.map.ObjectMapper;

import com.ebay.maui.helper.XMLHelper;
import com.google.common.base.Joiner;
import com.sun.jersey.api.client.Client;
import com.sun.jersey.api.client.ClientRequest;
import com.sun.jersey.api.client.ClientResponse;
import com.sun.jersey.api.client.UniformInterfaceException;
import com.sun.jersey.api.client.WebResource.Builder;
import com.sun.jersey.api.client.config.ClientConfig;
import com.sun.jersey.api.client.config.DefaultClientConfig;
import com.sun.jersey.api.json.JSONConfiguration;

/**
 * RestClient is for RESTful service testing based on Jersey client
 *
 * @author ehasenkamp
 */
public class RestClient<T> {
    private Client client;
    private Builder builder;
    private Class<T> model;

    private MediaType mediaType = MediaType.APPLICATION_JSON_TYPE;
    private Set<MediaType> accept = new HashSet<MediaType>();

    private Object postObject;
    private MediaType entityMediaType;

    private T response;

    private Properties headers;
    private ClientResponse clientResponse;
    private boolean continueOnBadResponse = false;

    public RestClient(Class<T> model) {
        init(model, null);
    }

    public RestClient(Class<T> model, Properties headers) {
        init(model, headers);
    }

    private void init(Class<T> model, Properties headers) {
        ClientConfig clientConfig = new DefaultClientConfig();
        clientConfig.getFeatures().put(JSONConfiguration.FEATURE_POJO_MAPPING,
                Boolean.TRUE);
        this.accept.add(MediaType.APPLICATION_JSON_TYPE);
        this.model = model;
        this.client = Client.create(clientConfig);
        this.client.addFilter(new RequestFilter());
        this.headers = headers;
    }
```



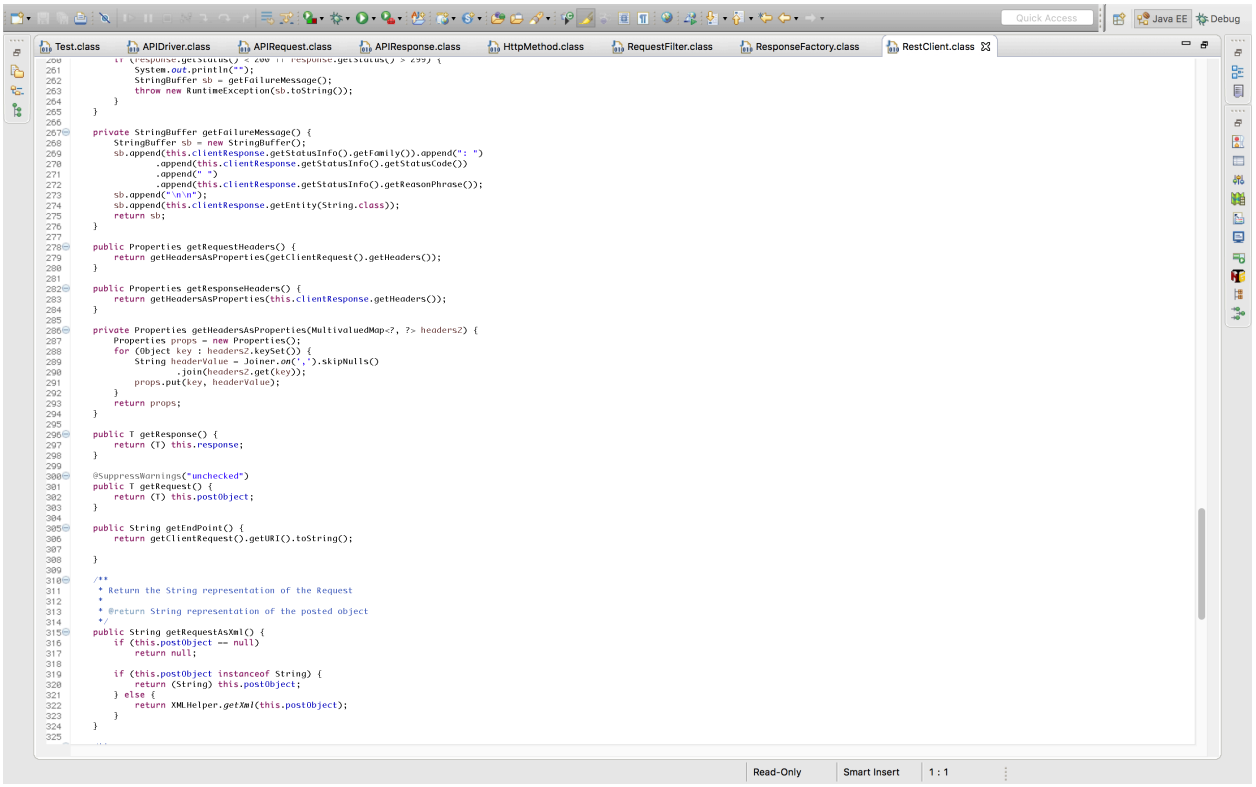

```java
    private void setBuilderDefaults(URL url) {
        this.builder = this.client.resource(url.toString()).getRequestBuilder();
        this.builder.type(this.mediaType);
        this.builder.accept(this.accept.toArray(new MediaType[this.accept
                .size()]));
        this.builder.entity(this.postObject, this.entityMediaType);
    }

    /**
     * Get the media type of the request
     *
     * @return media type
     */
    public MediaType getMediaType() {
        return this.mediaType;
    }

    /**
     * Set the media type of the request
     *
     * @param mediaType
     *            media type
     * @return a rest client
     */
    public RestClient<T> mediaType(MediaType mediaType) {
        this.mediaType = mediaType;
        return this;
    }

    /**
     * Set the accept types
     *
     * @param type
     *            media types
     * @return a rest client
     */
    public RestClient<T> accept(MediaType... type) {
        this.accept.addAll(new ArrayList<MediaType>(Arrays.asList(type)));
        return this;
    }

    /**
     * Set the headers
     *
     * @param name
     *            of the header
     * @return value of the header
     */
    public RestClient<T> header(String name, Object value) {
        if (this.headers == null)
            this.headers = new Properties();
        this.headers.put(name, value);
        return this;
    }

    /**
     * Set the request entity
     *
     * @param entity
     *            entity to be post
     * @return a rest client
     */
    public RestClient<T> entity(Object entity, MediaType mediaType) {
        this.postObject = entity;
        this.entityMediaType = mediaType;
        return this;
```



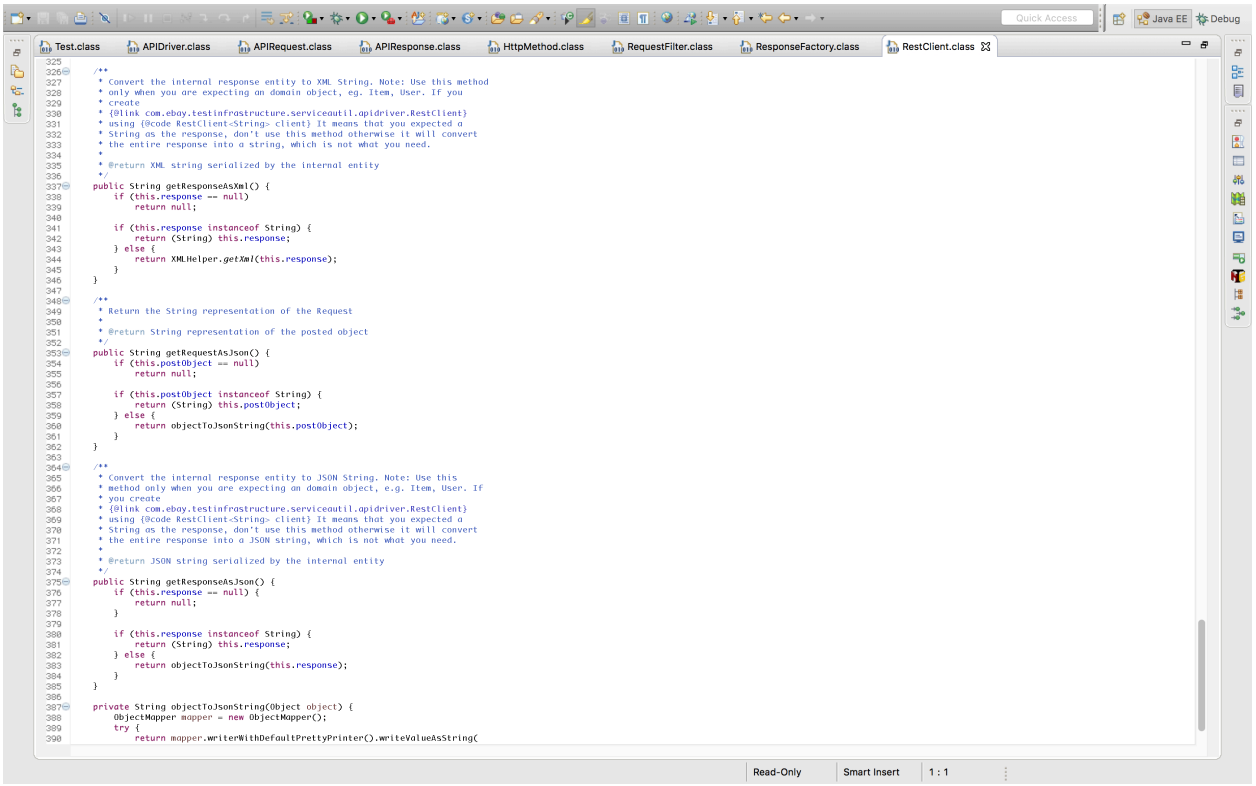

```
        return this;
    }

    /**
     * Use to test negative cases like checking for client error handling of the
     * service.
     *
     * @return {@link RestClient}
     */
    public RestClient<T> continueOnBadResponse() {
        this.continueOnBadResponse = true;
        return this;
    }

    /**
     * execute a GET method with the defined endpoint
     */
    public T get(URL url) {
        return get(url, null);
    }

    /**
     * execute a GET method with the defined endpoint and request object
     */
    public T get(URL url, Object entity) {
        return doCall(url, entity, HttpMethod.GET);
    }

    /**
     * execute a POST method with the defined endpoint
     */
    public T post(URL url) {
        return post(url, null);
    }

    /**
     * execute a POST method with the defined endpoint and request object
     */
    public T post(URL url, Object entity) {
        return doCall(url, entity, HttpMethod.POST);
    }

    /**
     * execute a PUT method with the defined endpoint
     */
    public T put(URL url) {
        return put(url, null);
    }

    /**
     * execute a PUT method with the defined endpoint and request object
     */
    public T put(URL url, Object entity) {
        return doCall(url, entity, HttpMethod.PUT);
    }

    /**
     * execute a DELETE method with the defined endpoint
     */
    public T delete(URL url) {
        return delete(url, null);
    }

    /**
     * execute a DELETE method with the defined endpoint and request object
     */
```




```
     * execute a DELETE method with the defined endpoint and request object
     */
    public T delete(URL url, Object entity) {
        return doCall(url, entity, HttpMethod.DELETE);
    }

    private T getReturnResponse(ClientResponse response) {
        this.response = response.getEntity(this.model);
        return this.response;
    }

    /**
     * @return the response code, for example 200, or 401, or 500, etc.
     */
    public int getResponseCode() {
        return this.clientResponse.getStatus();
    }

    private ClientResponse response(Object entity, HttpMethod httpMethod) {
        this.postObject = entity;
        this.clientResponse = ResponseFactory.getMethodType(this.builder,
                this.mediaType, entity, httpMethod, this.headers);
        return this.clientResponse;
    }

    @SuppressWarnings("unchecked")
    private T doCall(URL url, Object entity, HttpMethod httpMethod) {
        setBuilderDefaults(url);
        ClientResponse clientResponse = response(entity, httpMethod);
        logStatus(clientResponse);
        if (ClientResponse.class.equals(this.model)) {
            try {
                this.response = (T) clientResponse;
                new RestClientReportLogger().logRestAPIStep(this);
                return (T) clientResponse;
            } catch (UniformInterfaceException e) {
                return (T) e.getResponse();
            }
        } else {
            if (!continueOnBadResponse) {
                terminateOnBadResponse(clientResponse);
            }
            T response = getReturnResponse(clientResponse);
            new RestClientReportLogger().logRestAPIStep(this);
            return response;
        }
    }

    public String getHttpMethod() {
        return getClientRequest().getMethod();
    }

    private ClientRequest getClientRequest() {
        return (ClientRequest) this.clientResponse.getProperties().get(
                "REQUEST");
    }

    private void logStatus(ClientResponse response) {
        System.out.println("response.getStatus():"
                + response.getStatusInfo().getStatusCode());
    }

    private void terminateOnBadResponse(ClientResponse response) {
        if (response.getStatus() < 200 || response.getStatus() > 299) {
            System.out.println("");
            StringBuffer sb = getFailureMessage();
```

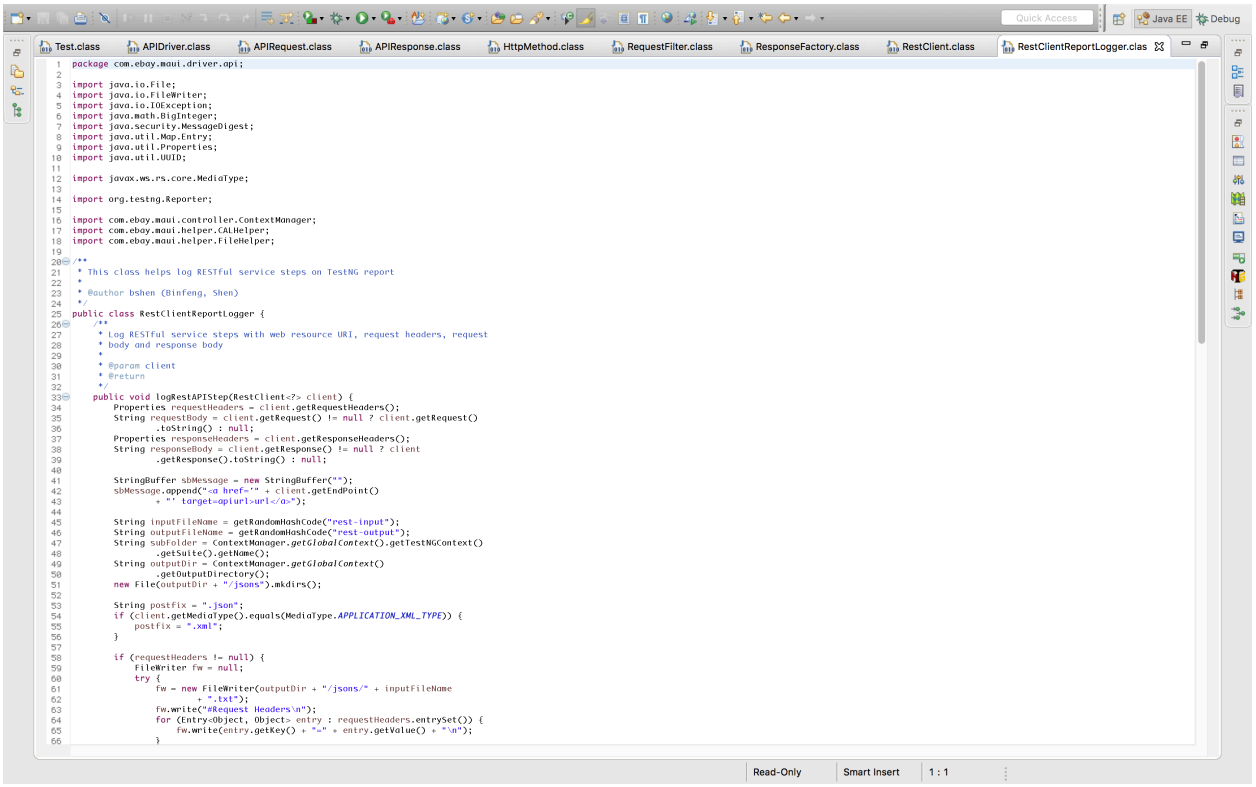


```
        if (this.response.getStatus() < 200 || this.response.getStatus() > 299) {
            System.out.println("");
            StringBuffer sb = getFailureMessage();
            throw new RuntimeException(sb.toString());
        }
    }

    private StringBuffer getFailureMessage() {
        StringBuffer sb = new StringBuffer();
        sb.append(this.clientResponse.getStatusInfo().getFamily()).append(": ")
                .append(this.clientResponse.getStatusInfo().getStatusCode())
                .append(" ")
                .append(this.clientResponse.getStatusInfo().getReasonPhrase());
        sb.append("\n\n");
        sb.append(this.clientResponse.getEntity(String.class));
        return sb;
    }

    public Properties getRequestHeaders() {
        return getHeadersAsProperties(getClientRequest().getHeaders());
    }

    public Properties getResponseHeaders() {
        return getHeadersAsProperties(this.clientResponse.getHeaders());
    }

    private Properties getHeadersAsProperties(MultivaluedMap<?, ?> headers2) {
        Properties props = new Properties();
        for (Object key : headers2.keySet()) {
            String headerValue = Joiner.on(',').skipNulls()
                    .join(headers2.get(key));
            props.put(key, headerValue);
        }
        return props;
    }

    public T getResponse() {
        return (T) this.response;
    }

    @SuppressWarnings("unchecked")
    public T getRequest() {
        return (T) this.postObject;
    }

    public String getEndPoint() {
        return getClientRequest().getURI().toString();

    }

    /**
     * Return the String representation of the Request
     *
     * @return String representation of the posted object
     */
    public String getRequestAsXml() {
        if (this.postObject == null)
            return null;

        if (this.postObject instanceof String) {
            return (String) this.postObject;
        } else {
            return XMLHelper.getXml(this.postObject);
        }
    }
```

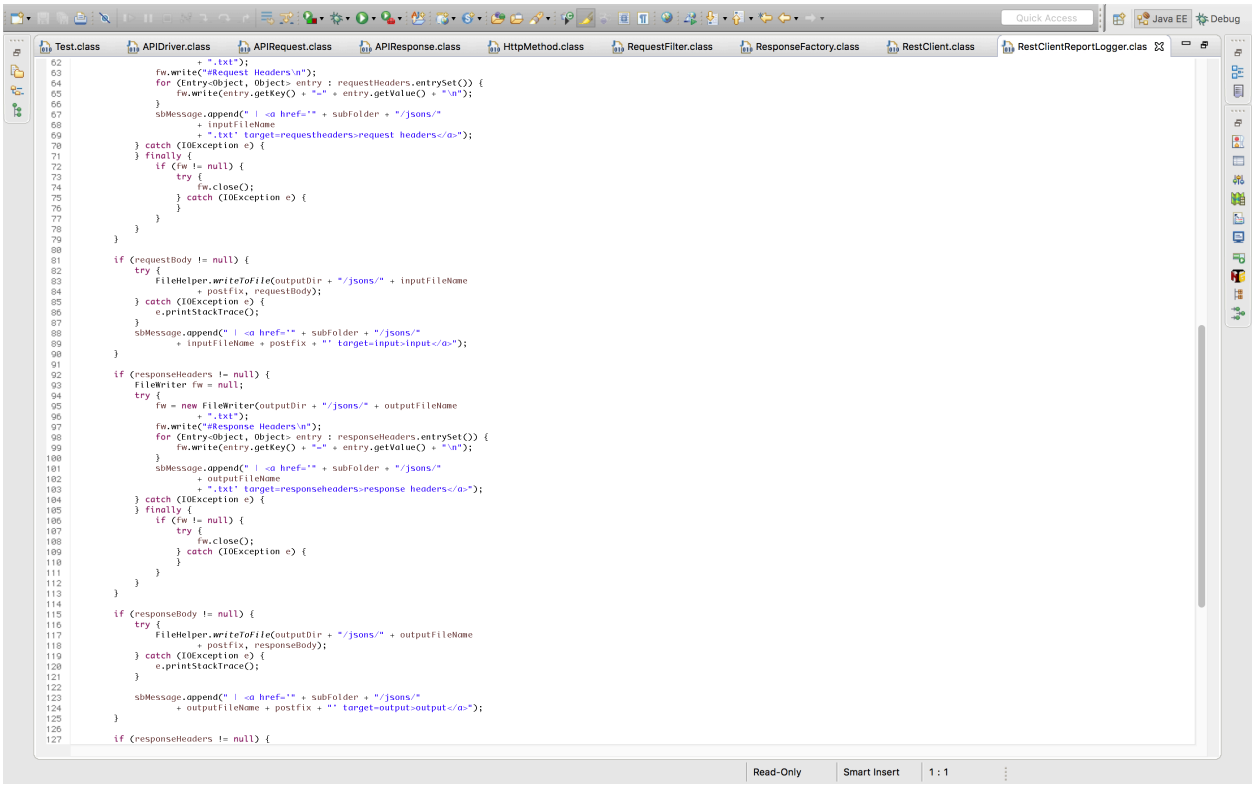


```
    /**
     * Convert the internal response entity to XML String. Note: Use this method
     * only when you are expecting an domain object, eg. Item, User. If you
     * create
     * {@link com.ebay.testinfrastructure.serviceautil.apidriver.RestClient}
     * using {@code RestClient<String> client} It means that you expected a
     * String as the response, don't use this method otherwise it will convert
     * the entire response into a string, which is not what you need.
     *
     * @return XML string serialized by the internal entity
     */
    public String getResponseAsXml() {
        if (this.response == null)
            return null;

        if (this.response instanceof String) {
            return (String) this.response;
        } else {
            return XMLHelper.getXml(this.response);
        }
    }

    /**
     * Return the String representation of the Request
     *
     * @return String representation of the posted object
     */
    public String getRequestAsJson() {
        if (this.postObject == null)
            return null;

        if (this.postObject instanceof String) {
            return (String) this.postObject;
        } else {
            return objectToJsonString(this.postObject);
        }
    }

    /**
     * Convert the internal response entity to JSON String. Note: Use this
     * method only when you are expecting an domain object, e.g. Item, User. If
     * you create
     * {@link com.ebay.testinfrastructure.serviceautil.apidriver.RestClient}
     * using {@code RestClient<String> client} It means that you expected a
     * String as the response, don't use this method otherwise it will convert
     * the entire response into a JSON string, which is not what you need.
     *
     * @return JSON string serialized by the internal entity
     */
    public String getResponseAsJson() {
        if (this.response == null) {
            return null;
        }

        if (this.response instanceof String) {
            return (String) this.response;
        } else {
            return objectToJsonString(this.response);
        }
    }

    private String objectToJsonString(Object object) {
        ObjectMapper mapper = new ObjectMapper();
        try {
            return mapper.writerWithDefaultPrettyPrinter().writeValueAsString(
```


```
     * using {@code RestClient<String> client} It means that you expected a
     * String as the response, don't use this method otherwise it will convert
     * the entire response into a string, which is not what you need.
     *
     * @return XML string serialized by the internal entity
     */
    public String getResponseAsXml() {
        if (this.response == null)
            return null;

        if (this.response instanceof String) {
            return (String) this.response;
        } else {
            return XMLHelper.getXml(this.response);
        }
    }

    /**
     * Return the String representation of the Request
     *
     * @return String representation of the posted object
     */
    public String getRequestAsJson() {
        if (this.postObject == null)
            return null;

        if (this.postObject instanceof String) {
            return (String) this.postObject;
        } else {
            return objectToJsonString(this.postObject);
        }
    }

    /**
     * Convert the internal response entity to JSON String. Note: Use this
     * method only when you are expecting an domain object, e.g. Item, User. If
     * you create
     * {@link com.ebay.testinfrastructure.serviceautil.apidriver.RestClient}
     * using {@code RestClient<String> client} It means that you expected a
     * String as the response, don't use this method otherwise it will convert
     * the entire response into a JSON string, which is not what you need.
     *
     * @return JSON string serialized by the internal entity
     */
    public String getResponseAsJson() {
        if (this.response == null) {
            return null;
        }

        if (this.response instanceof String) {
            return (String) this.response;
        } else {
            return objectToJsonString(this.response);
        }
    }

    private String objectToJsonString(Object object) {
        ObjectMapper mapper = new ObjectMapper();
        try {
            return mapper.writerWithDefaultPrettyPrinter().writeValueAsString(
                    object);
        } catch (IOException e) {
            throw new RuntimeException("Cannot convert object to JSON", e);
        }
    }
}
```

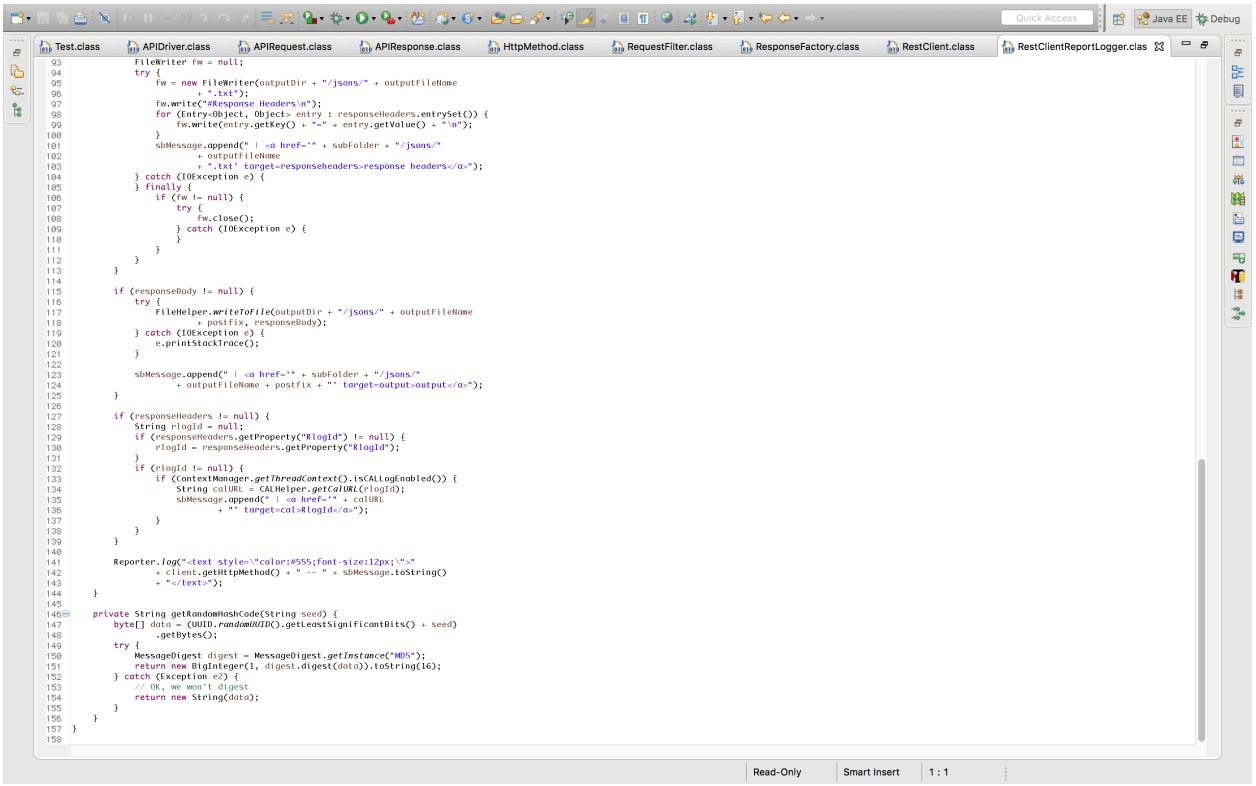



```
package com.ebay.maui.driver.api;

import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.math.BigInteger;
import java.security.MessageDigest;
import java.util.Map.Entry;
import java.util.Properties;
import java.util.UUID;

import javax.ws.rs.core.MediaType;

import org.testng.Reporter;

import com.ebay.maui.controller.ContextManager;
import com.ebay.maui.helper.CALHelper;
import com.ebay.maui.helper.FileHelper;

/**
 * This class helps log RESTful service steps on TestNG report
 *
 * @author bshen (Binfeng, Shen)
 */
public class RestClientReportLogger {
    /**
     * Log RESTful service steps with web resource URI, request headers, request
     * body and response body
     *
     * @param client
     * @return
     */
    public void logRestAPIStep(RestClient<?> client) {
        Properties requestHeaders = client.getRequestHeaders();
        String requestBody = client.getRequest() != null ? client.getRequest()
                .toString() : null;
        Properties responseHeaders = client.getResponseHeaders();
        String responseBody = client.getResponse() != null ? client
                .getResponse().toString() : null;

        StringBuffer sbMessage = new StringBuffer("");
        sbMessage.append("<a href='" + client.getEndPoint()
                + "' target=apiurl>url</a>");

        String inputFileName = getRandomHashCode("rest-input");
        String outputFileName = getRandomHashCode("rest-output");
        String subFolder = ContextManager.getGlobalContext().getTestNGContext()
                .getSuite().getName();
        String outputDir = ContextManager.getGlobalContext()
                .getOutputDirectory();
        new File(outputDir + "/jsons").mkdirs();

        String postfix = ".json";
        if (client.getMediaType().equals(MediaType.APPLICATION_XML_TYPE)) {
            postfix = ".xml";
        }

        if (requestHeaders != null) {
            FileWriter fw = null;
            try {
                fw = new FileWriter(outputDir + "/jsons/" + inputFileName
                        + ".txt");
                fw.write("#Request Headers\n");
                for (Entry<Object, Object> entry : requestHeaders.entrySet()) {
                    fw.write(entry.getKey() + "=" + entry.getValue() + "\n");
                }
```



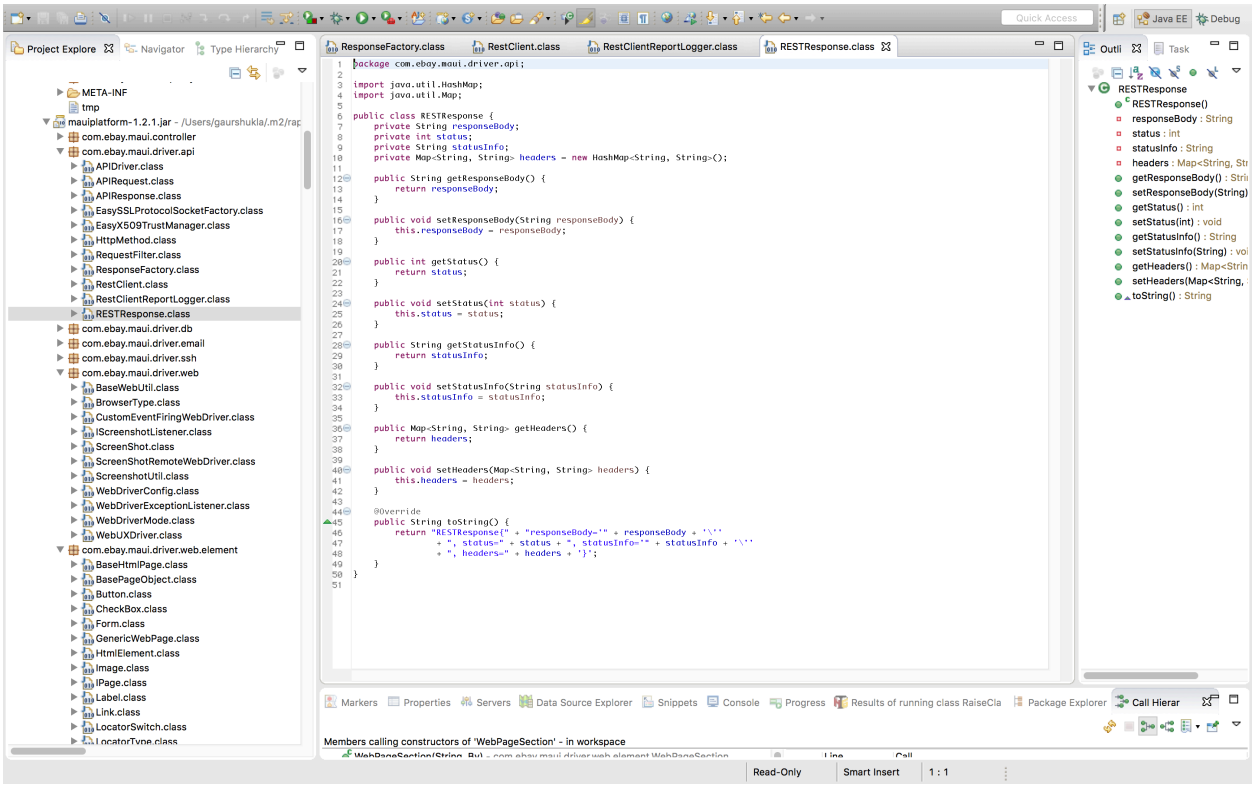
```
                + ".txt");
            fw.write("#Request Headers\n");
            for (Entry<Object, Object> entry : requestHeaders.entrySet()) {
                fw.write(entry.getKey() + "=" + entry.getValue() + "\n");
            }
            sbMessage.append(" | <a href='" + subFolder + "/jsons/"
                    + inputFileName
                    + ".txt' target=requestheaders>request headers</a>");
        } catch (IOException e) {
        } finally {
            if (fw != null) {
                try {
                    fw.close();
                } catch (IOException e) {
                }
            }
        }
    }

    if (requestBody != null) {
        try {
            FileHelper.writeToFile(outputDir + "/jsons/" + inputFileName
                    + postfix, requestBody);
        } catch (IOException e) {
            e.printStackTrace();
        }
        sbMessage.append(" | <a href='" + subFolder + "/jsons/"
                + inputFileName + postfix + "' target=input>input</a>");
    }

    if (responseHeaders != null) {
        FileWriter fw = null;
        try {
            fw = new FileWriter(outputDir + "/jsons/" + outputFileName
                    + ".txt");
            fw.write("#Response Headers\n");
            for (Entry<Object, Object> entry : responseHeaders.entrySet()) {
                fw.write(entry.getKey() + "=" + entry.getValue() + "\n");
            }
            sbMessage.append(" | <a href='" + subFolder + "/jsons/"
                    + outputFileName
                    + ".txt' target=responseheaders>response headers</a>");
        } catch (IOException e) {
        } finally {
            if (fw != null) {
                try {
                    fw.close();
                } catch (IOException e) {
                }
            }
        }
    }

    if (responseBody != null) {
        try {
            FileHelper.writeToFile(outputDir + "/jsons/" + outputFileName
                    + postfix, responseBody);
        } catch (IOException e) {
            e.printStackTrace();
        }

        sbMessage.append(" | <a href='" + subFolder + "/jsons/"
                + outputFileName + postfix + "' target=output>output</a>");
    }

    if (responseHeaders != null) {
```

Read-Only | Smart Insert | 1 : 1



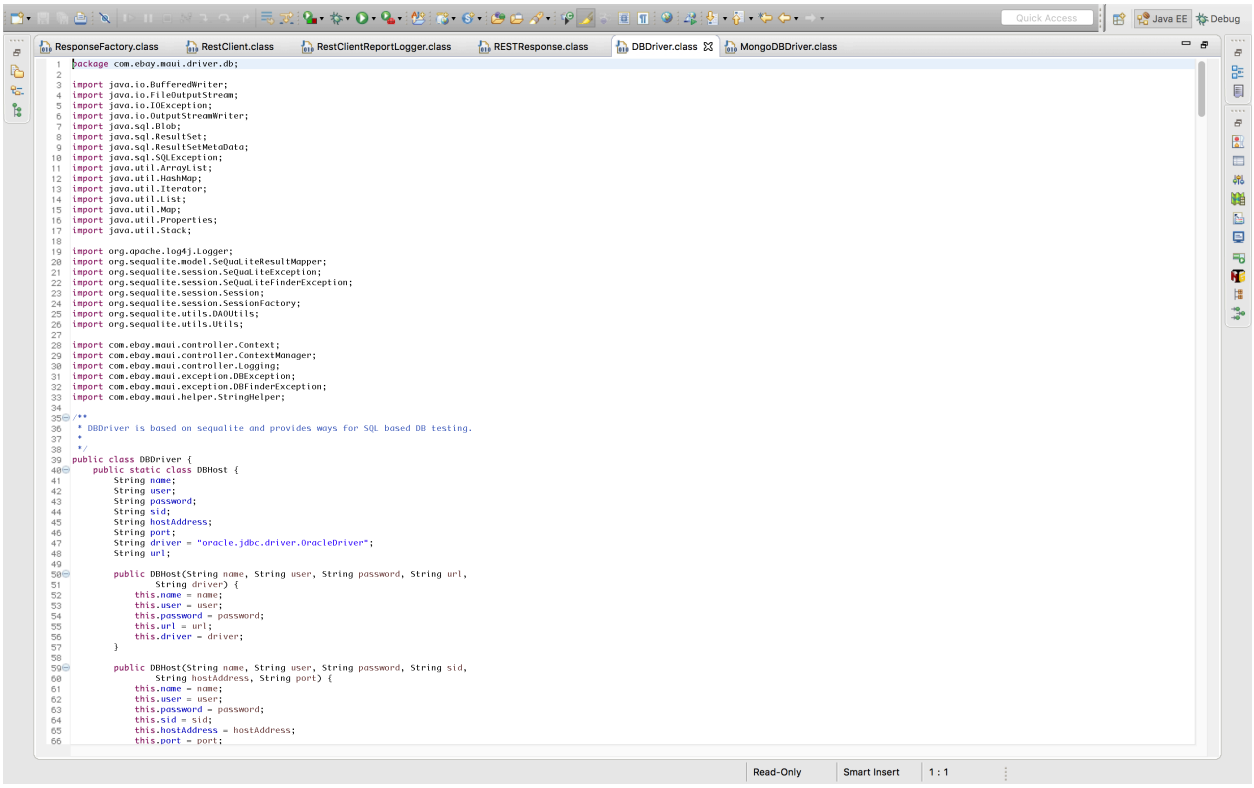
```
            FileWriter fw = null;
            try {
                fw = new FileWriter(outputDir + "/jsons/" + outputFileName
                        + ".txt");
                fw.write("#Response Headers\n");
                for (Entry<Object, Object> entry : responseHeaders.entrySet()) {
                    fw.write(entry.getKey() + "=" + entry.getValue() + "\n");
                }
                sbMessage.append(" | <a href='" + subFolder + "/jsons/"
                        + outputFileName
                        + ".txt' target=responseheaders>response headers</a>");
            } catch (IOException e) {
            } finally {
                if (fw != null) {
                    try {
                        fw.close();
                    } catch (IOException e) {
                    }
                }
            }
        }

        if (responseBody != null) {
            try {
                FileHelper.writeToFile(outputDir + "/jsons/" + outputFileName
                        + postfix, responseBody);
            } catch (IOException e) {
                e.printStackTrace();
            }

            sbMessage.append(" | <a href='" + subFolder + "/jsons/"
                    + outputFileName + postfix + "' target=output>output</a>");
        }

        if (responseHeaders != null) {
            String rlogId = null;
            if (responseHeaders.getProperty("RlogId") != null) {
                rlogId = responseHeaders.getProperty("RlogId");
            }
            if (rlogId != null) {
                if (ContextManager.getThreadContext().isCALLogEnabled()) {
                    String calURL = CALHelper.getCalURL(rlogId);
                    sbMessage.append(" | <a href='" + calURL
                            + "' target=cal>RlogId</a>");
                }
            }
        }

        Reporter.log("<text style=\"color:#555;font-size:12px;\">"
                + client.getHttpMethod() + " -- " + sbMessage.toString()
                + "</text>");
    }

    private String getRandomHashCode(String seed) {
        byte[] data = (UUID.randomUUID().getLeastSignificantBits() + seed)
                .getBytes();
        try {
            MessageDigest digest = MessageDigest.getInstance("MD5");
            return new BigInteger(1, digest.digest(data)).toString(16);
        } catch (Exception e2) {
            // OK, we won't digest
            return new String(data);
        }
    }
}
```

Read-Only | Smart Insert | 1 : 1

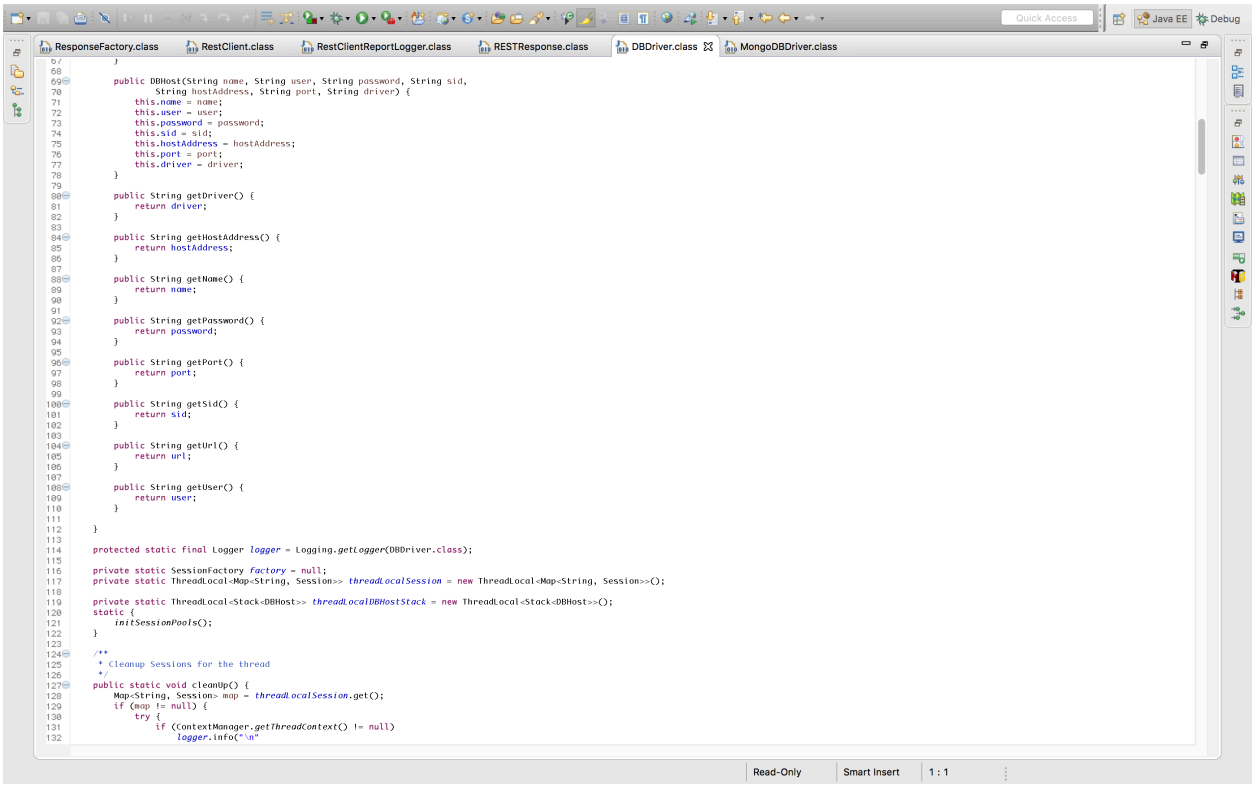



```java
package com.ebay.maui.driver.api;

import java.util.HashMap;
import java.util.Map;

public class RESTResponse {
    private String responseBody;
    private int status;
    private 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 + '}';
    }
}
```

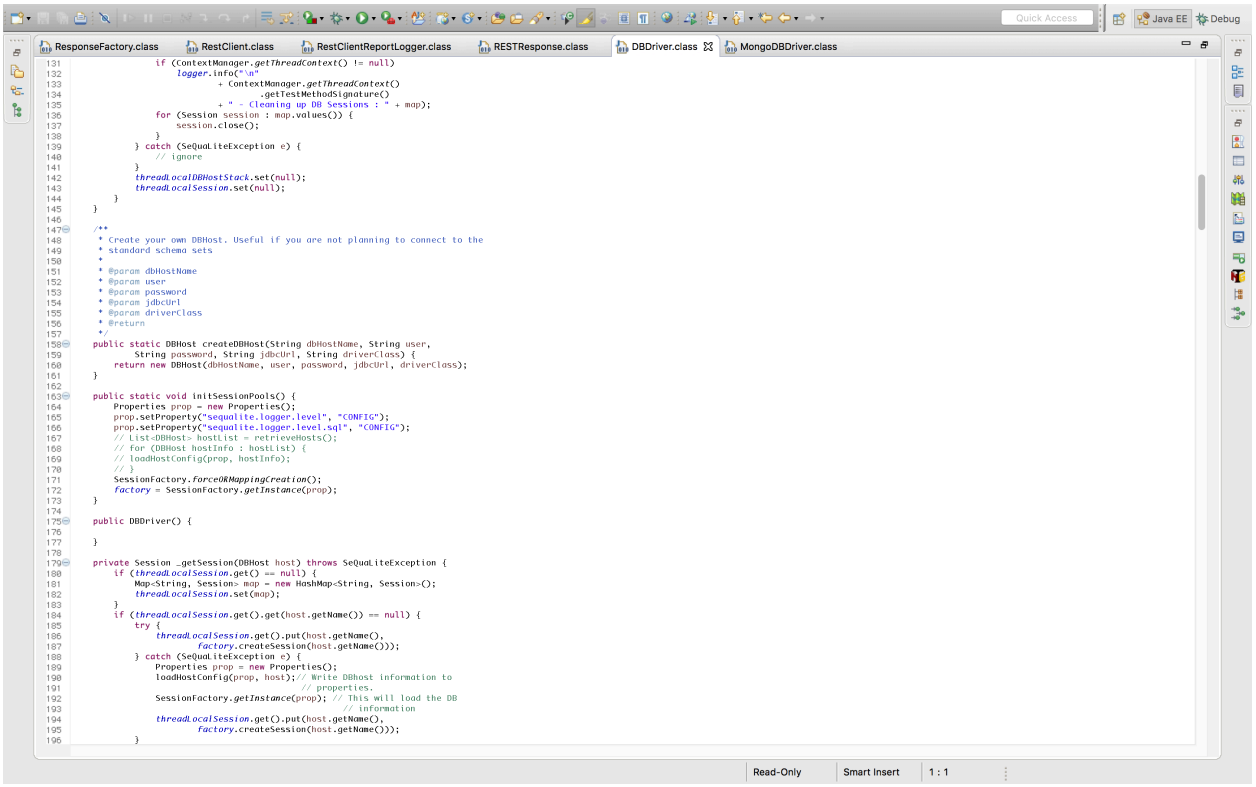
DB Driver:



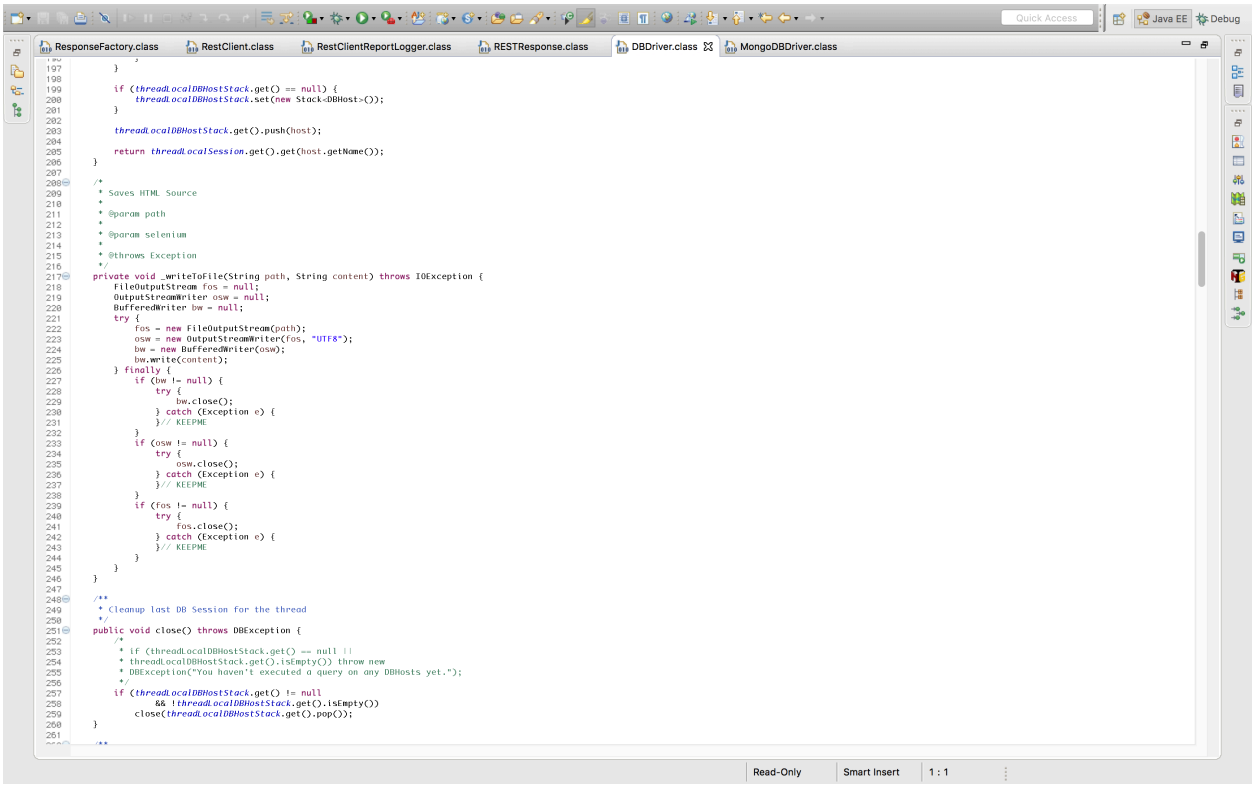

```
package com.ebay.maui.driver.db;

import java.io.BufferedWriter;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStreamWriter;
import java.sql.Blob;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import java.util.Properties;
import java.util.Stack;

import org.apache.log4j.Logger;
import org.sequalite.model.SeQuaLiteResultMapper;
import org.sequalite.session.SeQuaLiteException;
import org.sequalite.session.SeQuaLiteFinderException;
import org.sequalite.session.Session;
import org.sequalite.session.SessionFactory;
import org.sequalite.utils.DAOUtils;
import org.sequalite.utils.Utils;

import com.ebay.maui.controller.Context;
import com.ebay.maui.controller.ContextManager;
import com.ebay.maui.controller.Logging;
import com.ebay.maui.exception.DBException;
import com.ebay.maui.exception.DBFinderException;
import com.ebay.maui.helper.StringHelper;

/**
 * DBDriver is based on sequalite and provides ways for SQL based DB testing.
 *
 */
public class DBDriver {
    public static class DBHost {
        String name;
        String user;
        String password;
        String sid;
        String hostAddress;
        String port;
        String driver = "oracle.jdbc.driver.OracleDriver";
        String url;

        public DBHost(String name, String user, String password, String url,
                String driver) {
            this.name = name;
            this.user = user;
            this.password = password;
            this.url = url;
            this.driver = driver;
        }

        public DBHost(String name, String user, String password, String sid,
                String hostAddress, String port) {
            this.name = name;
            this.user = user;
            this.password = password;
            this.sid = sid;
            this.hostAddress = hostAddress;
            this.port = port;
```



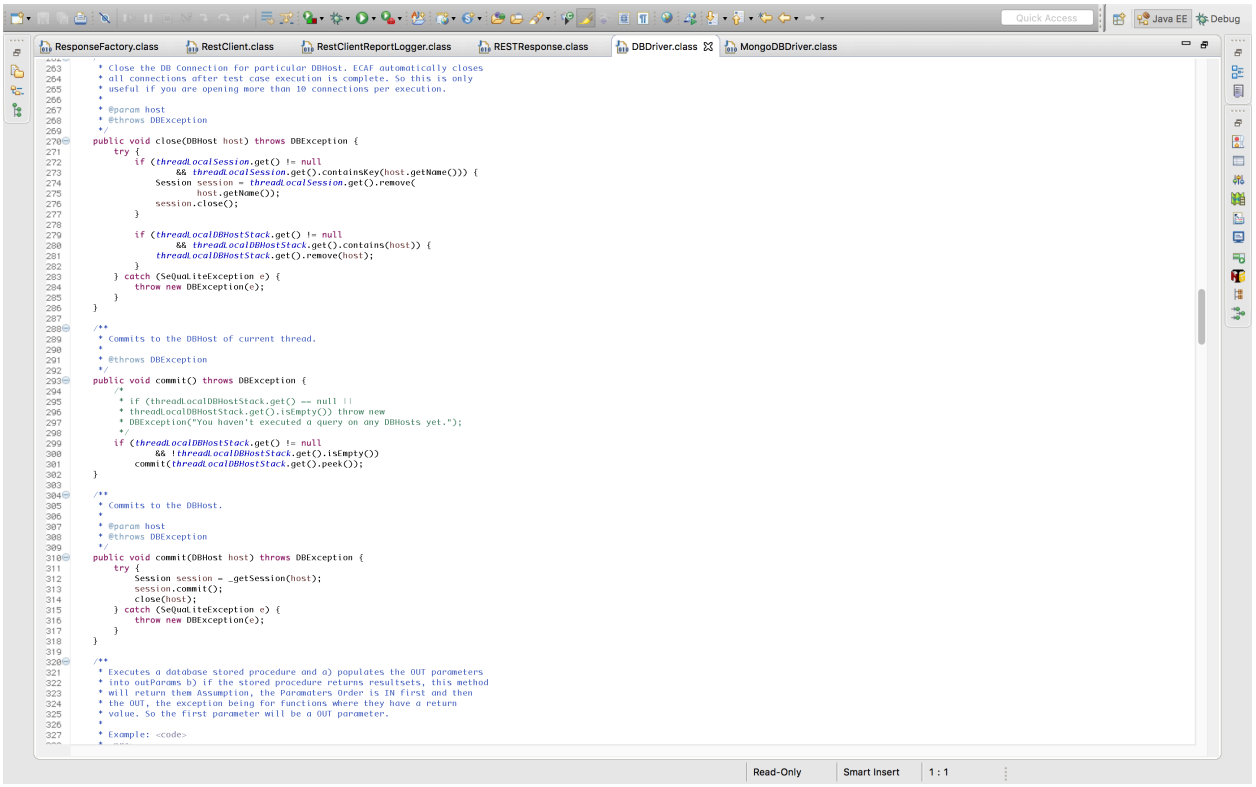

```
        }

        public DBHost(String name, String user, String password, String sid,
                String hostAddress, String port, String driver) {
            this.name = name;
            this.user = user;
            this.password = password;
            this.sid = sid;
            this.hostAddress = hostAddress;
            this.port = port;
            this.driver = driver;
        }

        public String getDriver() {
            return driver;
        }

        public String getHostAddress() {
            return hostAddress;
        }

        public String getName() {
            return name;
        }

        public String getPassword() {
            return password;
        }

        public String getPort() {
            return port;
        }

        public String getSid() {
            return sid;
        }

        public String getUrl() {
            return url;
        }

        public String getUser() {
            return user;
        }

    }

    protected static final Logger logger = Logging.getLogger(DBDriver.class);

    private static SessionFactory factory = null;
    private static ThreadLocal<Map<String, Session>> threadLocalSession = new ThreadLocal<Map<String, Session>>();

    private static ThreadLocal<Stack<DBHost>> threadLocalDBHostStack = new ThreadLocal<Stack<DBHost>>();
    static {
        initSessionPools();
    }

    /**
     * Cleanup Sessions for the thread
     */
    public static void cleanUp() {
        Map<String, Session> map = threadLocalSession.get();
        if (map != null) {
            try {
                if (ContextManager.getThreadContext() != null)
                    logger.info("\n"
```

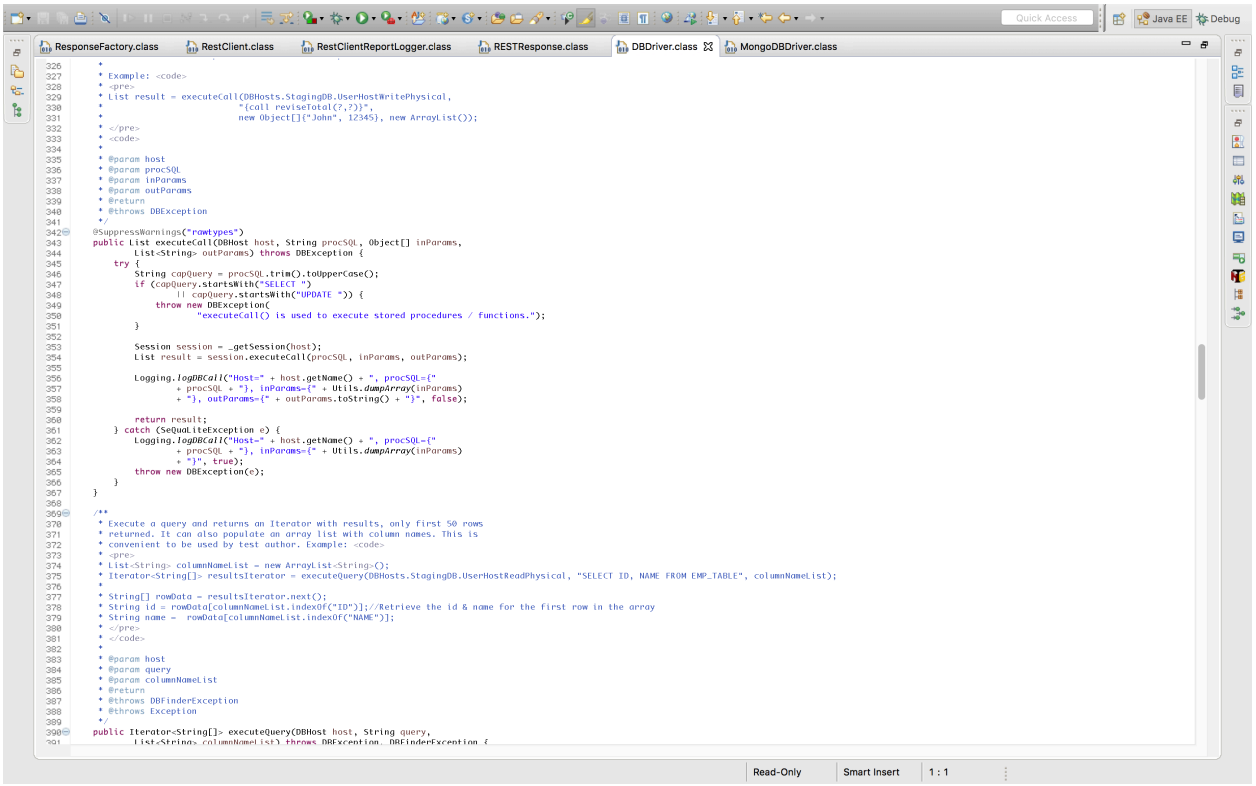


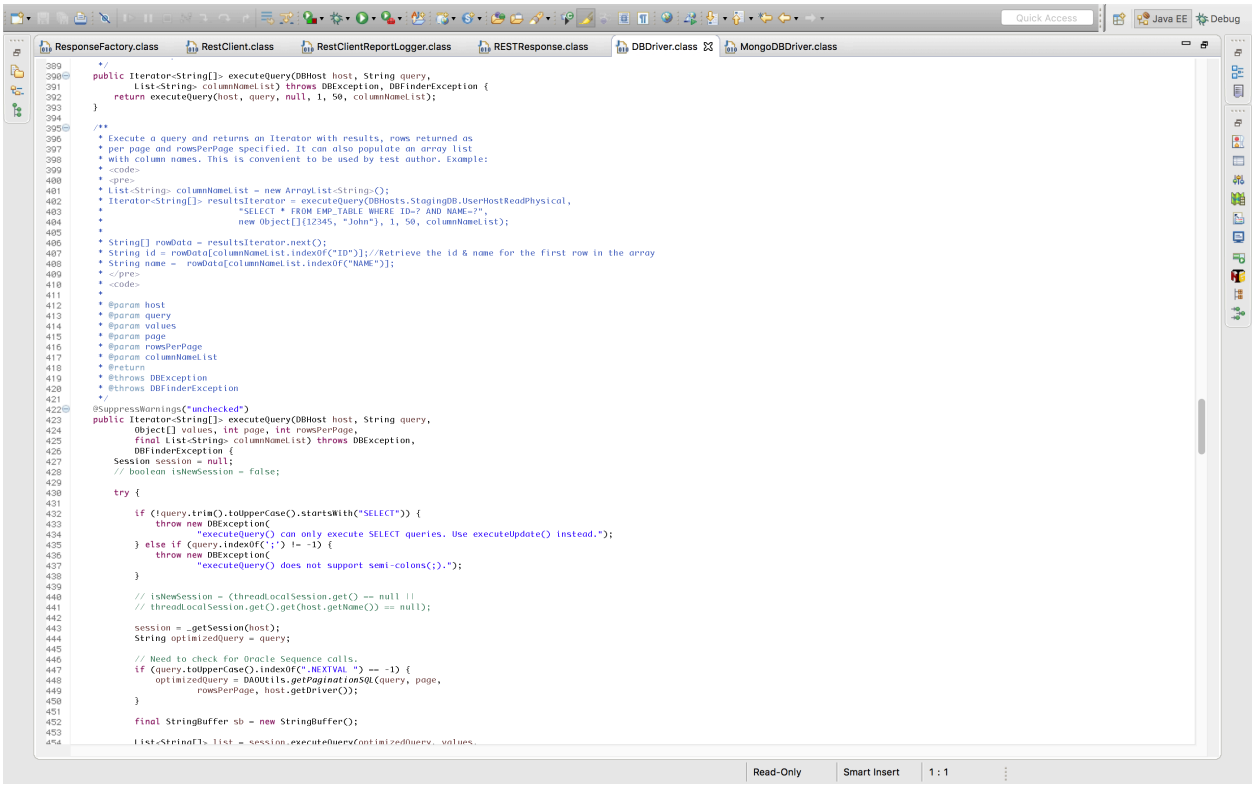

```
                if (ContextManager.getThreadContext() != null)
                    logger.info("\n"
                            + ContextManager.getThreadContext()
                                    .getTestMethodSignature()
                            + " - Cleaning up DB Sessions : " + map);
                for (Session session : map.values()) {
                    session.close();
                }
            } catch (SeQuaLiteException e) {
                // ignore
            }
            threadLocalDBHostStack.set(null);
            threadLocalSession.set(null);
        }
    }

    /**
     * Create your own DBHost. Useful if you are not planning to connect to the
     * standard schema sets
     *
     * @param dbHostName
     * @param user
     * @param password
     * @param jdbcUrl
     * @param driverClass
     * @return
     */
    public static DBHost createDBHost(String dbHostName, String user,
            String password, String jdbcUrl, String driverClass) {
        return new DBHost(dbHostName, user, password, jdbcUrl, driverClass);
    }

    public static void initSessionPools() {
        Properties prop = new Properties();
        prop.setProperty("sequalite.logger.level", "CONFIG");
        prop.setProperty("sequalite.logger.level.sql", "CONFIG");
        // List<DBHost> hostList = retrieveHosts();
        // for (DBHost hostInfo : hostList) {
        // loadHostConfig(prop, hostInfo);
        // }
        SessionFactory.forceORMappingCreation();
        factory = SessionFactory.getInstance(prop);
    }

    public DBDriver() {

    }

    private Session _getSession(DBHost host) throws SeQuaLiteException {
        if (threadLocalSession.get() == null) {
            Map<String, Session> map = new HashMap<String, Session>();
            threadLocalSession.set(map);
        }
        if (threadLocalSession.get().get(host.getName()) == null) {
            try {
                threadLocalSession.get().put(host.getName(),
                        factory.createSession(host.getName()));
            } catch (SeQuaLiteException e) {
                Properties prop = new Properties();
                loadHostConfig(prop, host);// Write DBhost information to
                                           // properties.
                SessionFactory.getInstance(prop); // This will load the DB
                                                  // information
                threadLocalSession.get().put(host.getName(),
                        factory.createSession(host.getName()));
            }
```



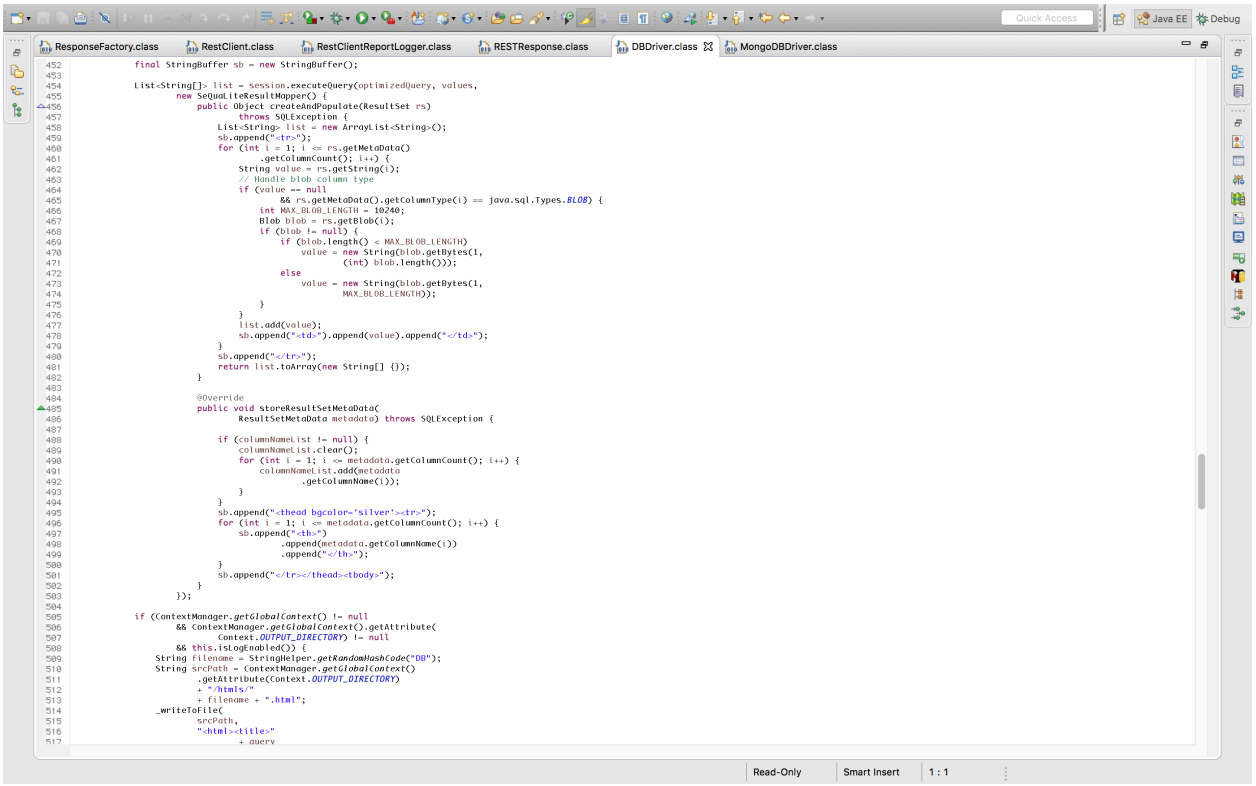

```
        }

        if (threadLocalDBHostStack.get() == null) {
            threadLocalDBHostStack.set(new Stack<DBHost>());
        }

        threadLocalDBHostStack.get().push(host);

        return threadLocalSession.get().get(host.getName());
    }

    /*
     * Saves HTML Source
     *
     * @param path
     *
     * @param selenium
     *
     * @throws Exception
     */
    private void _writeToFile(String path, String content) throws IOException {
        FileOutputStream fos = null;
        OutputStreamWriter osw = null;
        BufferedWriter bw = null;
        try {
            fos = new FileOutputStream(path);
            osw = new OutputStreamWriter(fos, "UTF8");
            bw = new BufferedWriter(osw);
            bw.write(content);
        } finally {
            if (bw != null) {
                try {
                    bw.close();
                } catch (Exception e) {
                }// KEEPME
            }
            if (osw != null) {
                try {
                    osw.close();
                } catch (Exception e) {
                }// KEEPME
            }
            if (fos != null) {
                try {
                    fos.close();
                } catch (Exception e) {
                }// KEEPME
            }
        }
    }

    /**
     * Cleanup last DB Session for the thread
     */
    public void close() throws DBException {
        /*
         * if (threadLocalDBHostStack.get() == null ||
         * threadLocalDBHostStack.get().isEmpty()) throw new
         * DBException("You haven't executed a query on any DBHosts yet.");
         */
        if (threadLocalDBHostStack.get() != null
                && !threadLocalDBHostStack.get().isEmpty())
            close(threadLocalDBHostStack.get().pop());
    }

    /**
```


ResponseFactory.class
RestClient.class
RestClientReportLogger.class
RESTResponse.class
DBDriver.class
MongoDBDriver.class
Quick Access
Java EE
Debug
Read-Only
Smart Insert
1 : 1

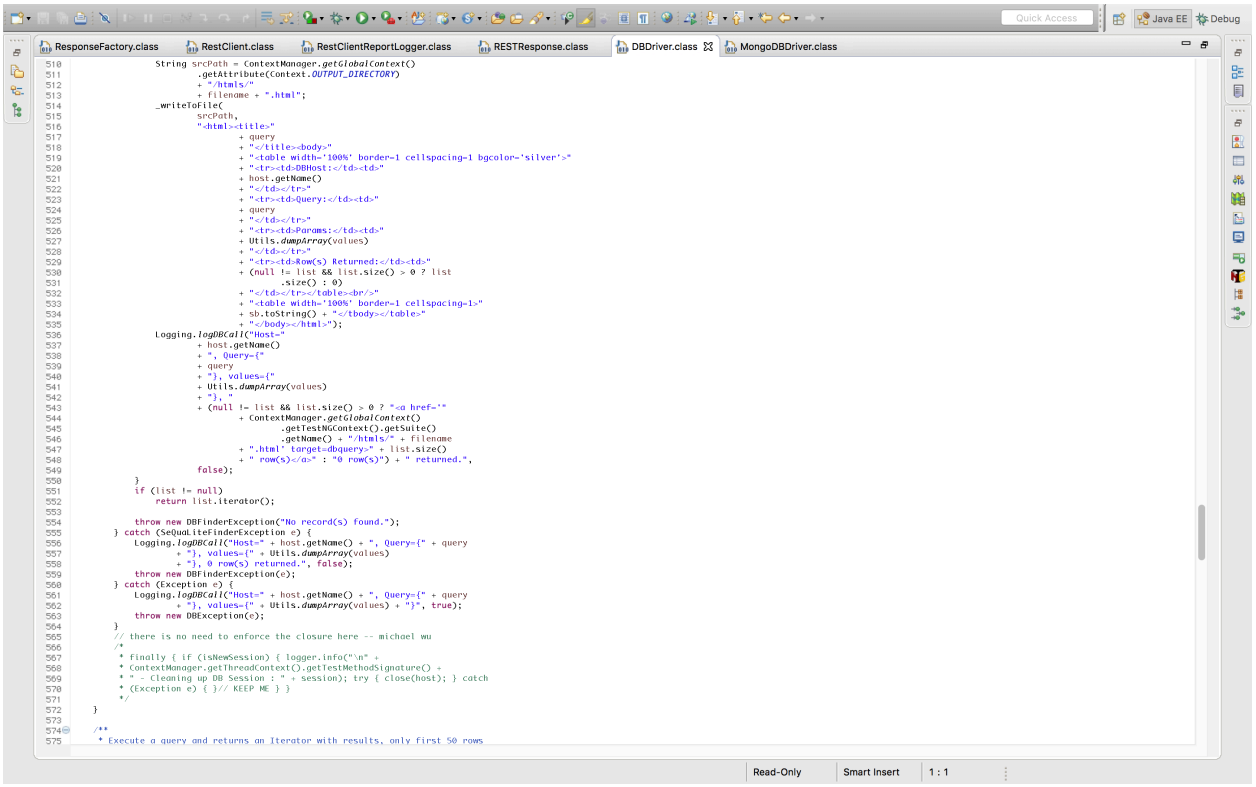

ResponseFactory.class
RestClient.class
RestClientReportLogger.class
RESTResponse.class
DBDriver.class
MongoDBDriver.class
Quick Access
Java EE
Debug
Read-Only
Smart Insert
1 : 1

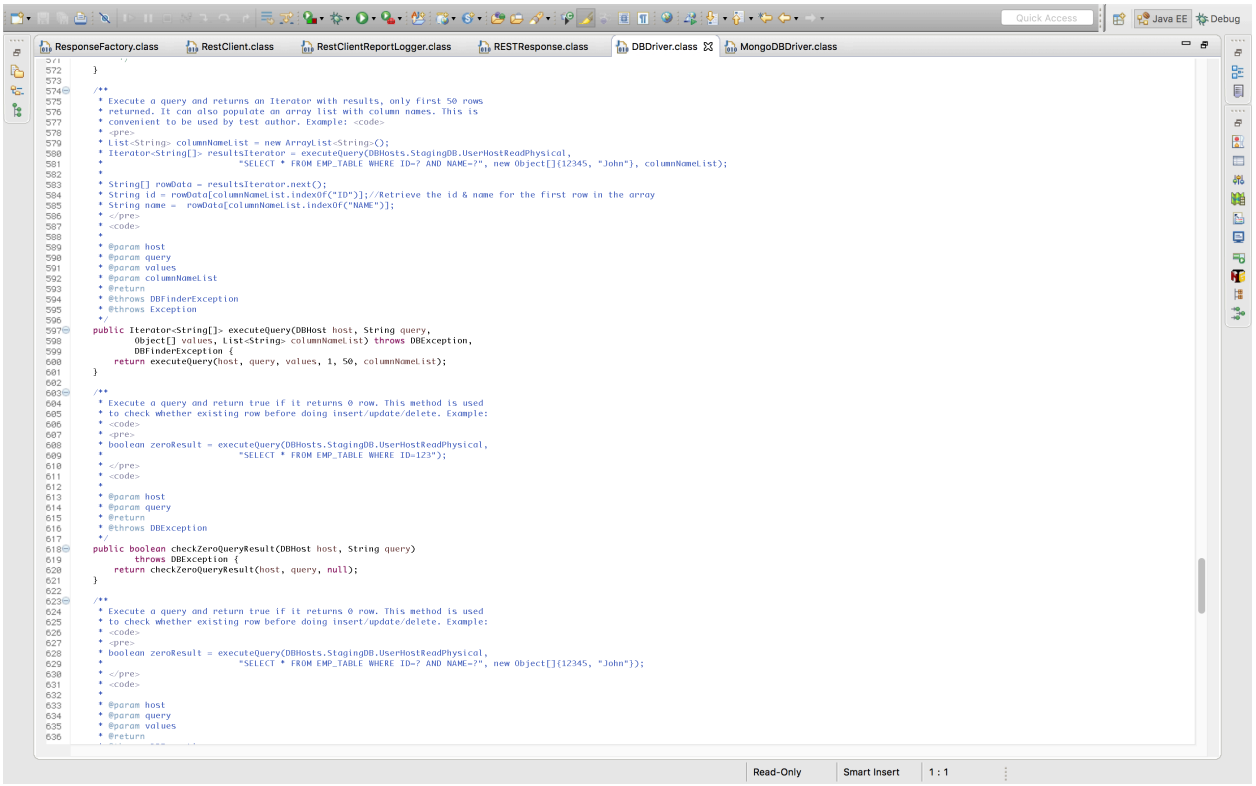


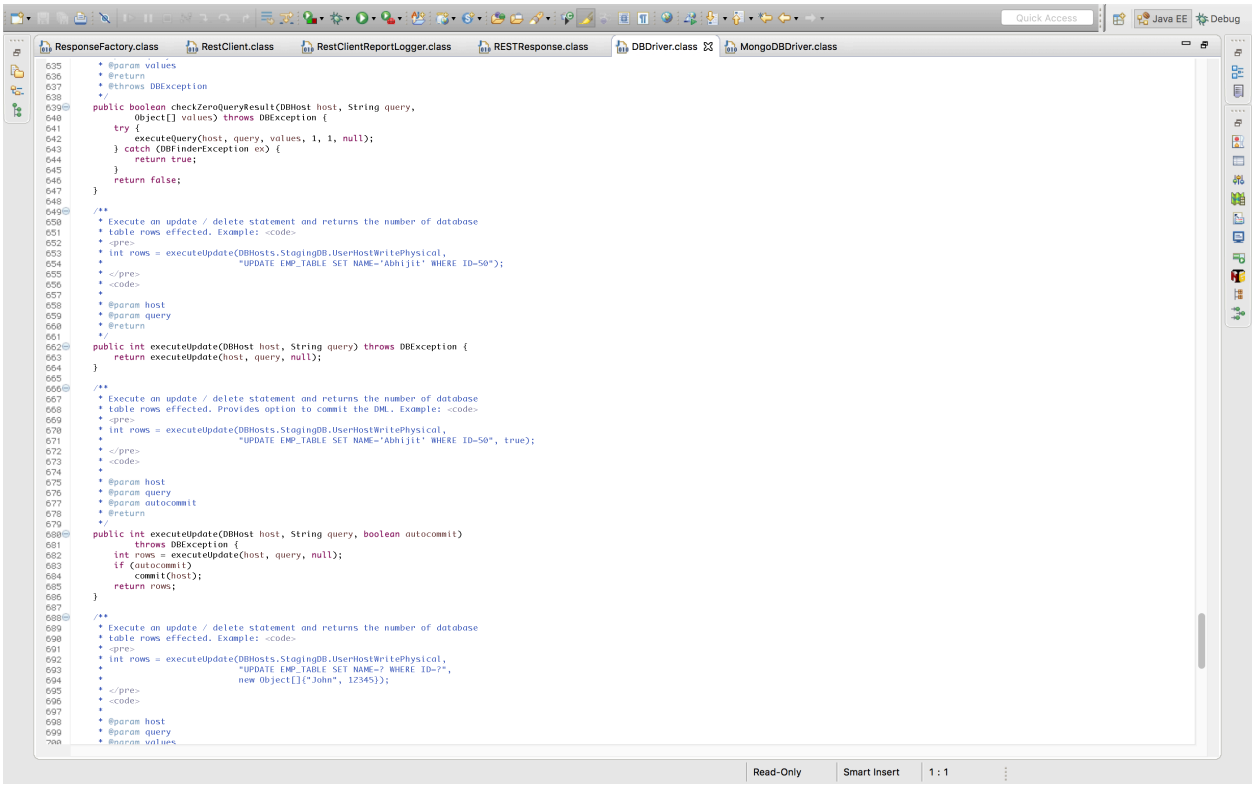

```
     */
    public Iterator<String[]> executeQuery(DBHost host, String query,
            List<String> columnNameList) throws DBException, DBFinderException {
        return executeQuery(host, query, null, 1, 50, columnNameList);
    }

    /**
     * Execute a query and returns an Iterator with results, rows returned as
     * per page and rowsPerPage specified. It can also populate an array list
     * with column names. This is convenient to be used by test author. Example:
     * <code>
     * <pre>
     * List<String> columnNameList = new ArrayList<String>();
     * Iterator<String[]> resultsIterator = executeQuery(DBHosts.StagingDB.UserHostReadPhysical,
     *                         "SELECT * FROM EMP_TABLE WHERE ID=? AND NAME=?",
     *                         new Object[]{12345, "John"}, 1, 50, columnNameList);
     *
     * String[] rowData = resultsIterator.next();
     * String id = rowData[columnNameList.indexOf("ID")];//Retrieve the id & name for the first row in the array
     * String name =  rowData[columnNameList.indexOf("NAME")];
     * </pre>
     * </code>
     *
     * @param host
     * @param query
     * @param values
     * @param page
     * @param rowsPerPage
     * @param columnNameList
     * @return
     * @throws DBException
     * @throws DBFinderException
     */
    @SuppressWarnings("unchecked")
    public Iterator<String[]> executeQuery(DBHost host, String query,
            Object[] values, int page, int rowsPerPage,
            final List<String> columnNameList) throws DBException,
            DBFinderException {
        Session session = null;
        // boolean isNewSession = false;

        try {

            if (!query.trim().toUpperCase().startsWith("SELECT")) {
                throw new DBException(
                        "executeQuery() can only execute SELECT queries. Use executeUpdate() instead.");
            } else if (query.indexOf(';') != -1) {
                throw new DBException(
                        "executeQuery() does not support semi-colons(;).");
            }

            // isNewSession = (threadLocalSession.get() == null ||
            // threadLocalSession.get().get(host.getName()) == null);

            session = _getSession(host);
            String optimizedQuery = query;

            // Need to check for Oracle Sequence calls.
            if (query.toUpperCase().indexOf(".NEXTVAL ") == -1) {
                optimizedQuery = DAOUtils.getPaginationSQL(query, page,
                        rowsPerPage, host.getDriver());
            }

            final StringBuffer sb = new StringBuffer();

            List<String[]> list = session.executeQuery(optimizedQuery, values,
```



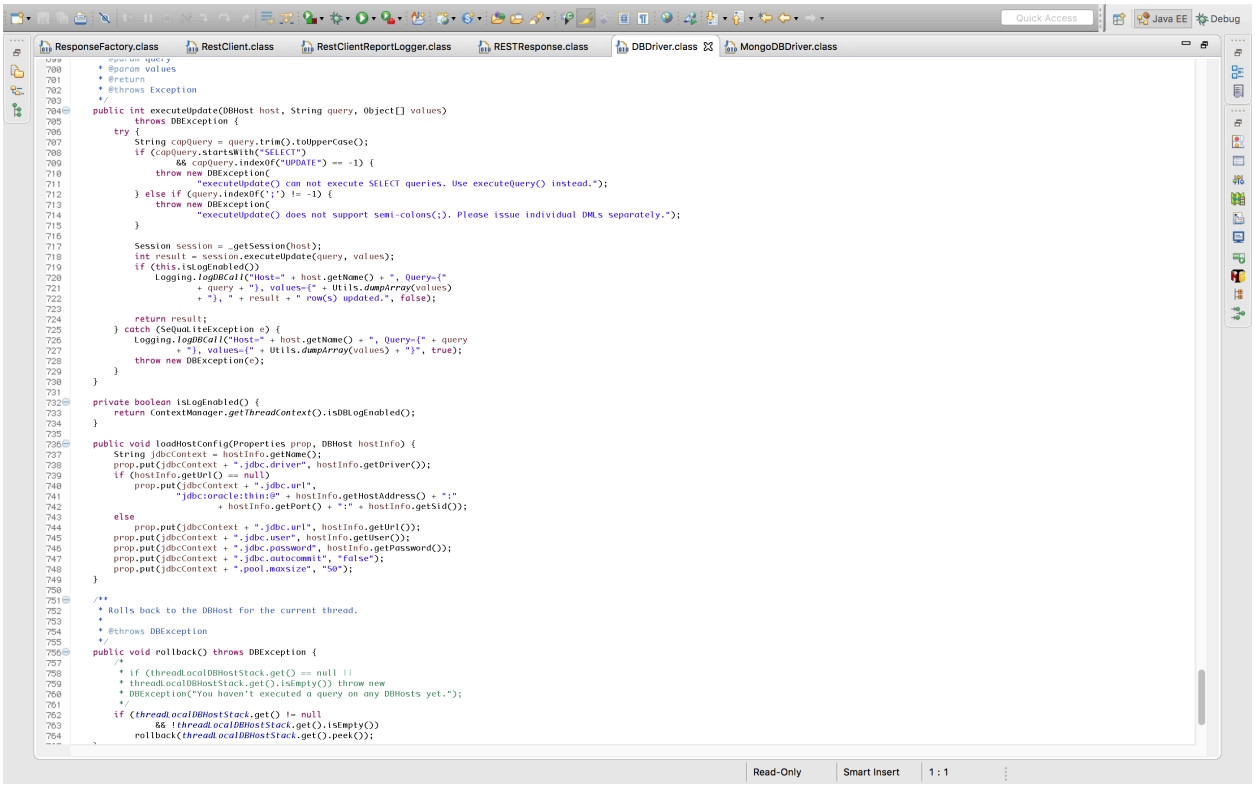

```
            final StringBuffer sb = new StringBuffer();

            List<String[]> list = session.executeQuery(optimizedQuery, values,
                    new SeQuaLiteResultMapper() {
                        public Object createAndPopulate(ResultSet rs)
                                throws SQLException {
                            List<String> list = new ArrayList<String>();
                            sb.append("<tr>");
                            for (int i = 1; i <= rs.getMetaData()
                                    .getColumnCount(); i++) {
                                String value = rs.getString(i);
                                // Handle blob column type
                                if (value == null
                                        && rs.getMetaData().getColumnType(i) == java.sql.Types.BLOB) {
                                    int MAX_BLOB_LENGTH = 10240;
                                    Blob blob = rs.getBlob(i);
                                    if (blob != null) {
                                        if (blob.length() < MAX_BLOB_LENGTH)
                                            value = new String(blob.getBytes(1,
                                                    (int) blob.length()));
                                        else
                                            value = new String(blob.getBytes(1,
                                                    MAX_BLOB_LENGTH));
                                    }
                                }
                                list.add(value);
                                sb.append("<td>").append(value).append("</td>");
                            }
                            sb.append("</tr>");
                            return list.toArray(new String[] {});
                        }

                        @Override
                        public void storeResultSetMetaData(
                                ResultSetMetaData metadata) throws SQLException {

                            if (columnNameList != null) {
                                columnNameList.clear();
                                for (int i = 1; i <= metadata.getColumnCount(); i++) {
                                    columnNameList.add(metadata
                                            .getColumnName(i));
                                }
                            }
                            sb.append("<thead bgcolor='silver'><tr>");
                            for (int i = 1; i <= metadata.getColumnCount(); i++) {
                                sb.append("<th>")
                                        .append(metadata.getColumnName(i))
                                        .append("</th>");
                            }
                            sb.append("</tr></thead><tbody>");
                        }
                    });

            if (ContextManager.getGlobalContext() != null
                    && ContextManager.getGlobalContext().getAttribute(
                            Context.OUTPUT_DIRECTORY) != null
                    && this.isLogEnabled()) {
                String filename = StringHelper.getRandomHashCode("DB");
                String srcPath = ContextManager.getGlobalContext()
                        .getAttribute(Context.OUTPUT_DIRECTORY)
                        + "/html5/"
                        + filename + ".html";
                _writeToFile(
                        srcPath,
                        "<html><title>"
                                + query
```


ResponseFactory.class
RestClient.class
RestClientReportLogger.class
RESTResponse.class
DBDriver.class
MongoDBDriver.class
Quick Access
Java EE
Debug
Read-Only
Smart Insert
1 : 1

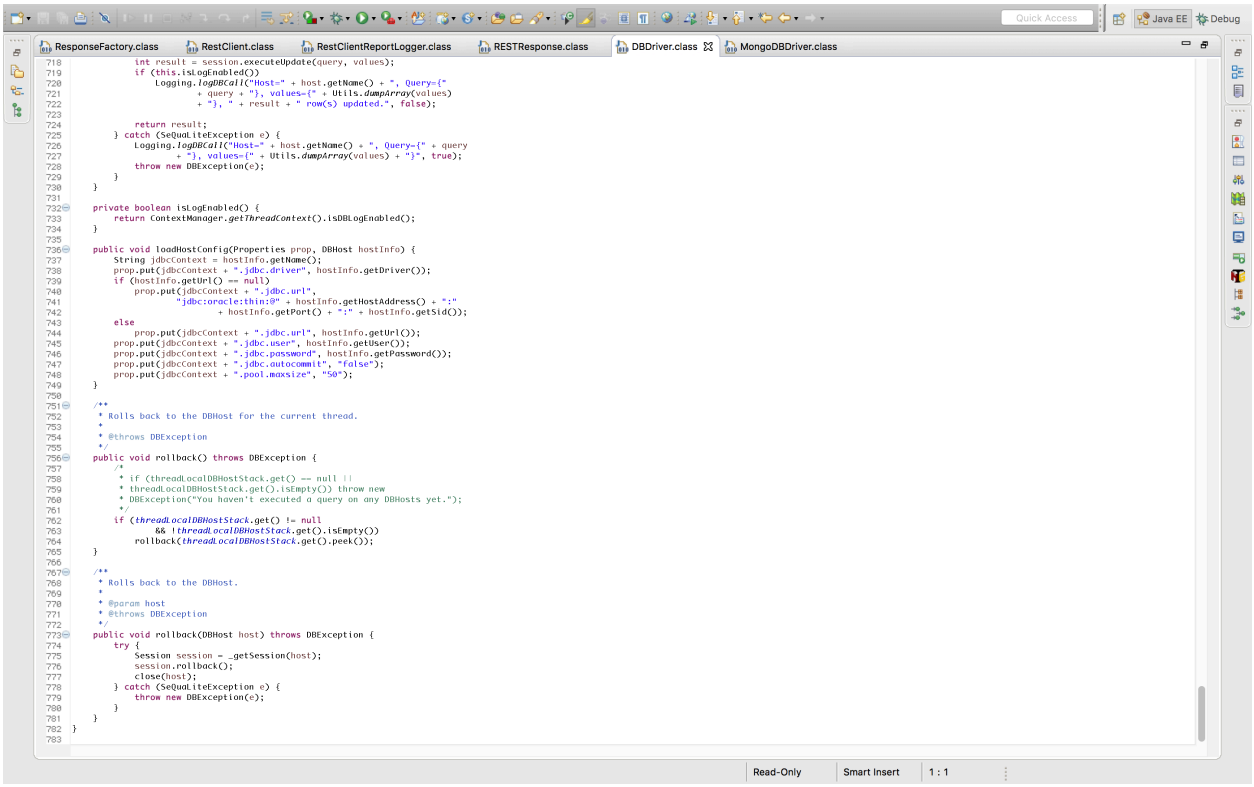

ResponseFactory.class
RestClient.class
RestClientReportLogger.class
RESTResponse.class
DBDriver.class
MongoDBDriver.class
Quick Access
Java EE
Debug
Read-Only
Smart Insert
1 : 1




```
     * @param values
     * @return
     * @throws DBException
     */
    public boolean checkZeroQueryResult(DBHost host, String query,
            Object[] values) throws DBException {
        try {
            executeQuery(host, query, values, 1, 1, null);
        } catch (DBFinderException ex) {
            return true;
        }
        return false;
    }

    /**
     * Execute an update / delete statement and returns the number of database
     * table rows effected. Example: <code>
     * <pre>
     * int rows = executeUpdate(DBHosts.StagingDB.UserHostWritePhysical,
     *                          "UPDATE EMP_TABLE SET NAME='Abhijit' WHERE ID=50");
     * </pre>
     * <code>
     *
     * @param host
     * @param query
     * @return
     */
    public int executeUpdate(DBHost host, String query) throws DBException {
        return executeUpdate(host, query, null);
    }

    /**
     * Execute an update / delete statement and returns the number of database
     * table rows effected. Provides option to commit the DML. Example: <code>
     * <pre>
     * int rows = executeUpdate(DBHosts.StagingDB.UserHostWritePhysical,
     *                          "UPDATE EMP_TABLE SET NAME='Abhijit' WHERE ID=50", true);
     * </pre>
     * <code>
     *
     * @param host
     * @param query
     * @param autocommit
     * @return
     */
    public int executeUpdate(DBHost host, String query, boolean autocommit)
            throws DBException {
        int rows = executeUpdate(host, query, null);
        if (autocommit)
            commit(host);
        return rows;
    }

    /**
     * Execute an update / delete statement and returns the number of database
     * table rows effected. Example: <code>
     * <pre>
     * int rows = executeUpdate(DBHosts.StagingDB.UserHostWritePhysical,
     *                          "UPDATE EMP_TABLE SET NAME=? WHERE ID=?",
     *                          new Object[]{"John", 12345});
     * </pre>
     * <code>
     *
     * @param host
     * @param query
     * @param values
```




```
     * @param values
     * @return
     * @throws Exception
     */
    public int executeUpdate(DBHost host, String query, Object[] values)
            throws DBException {
        try {
            String capQuery = query.trim().toUpperCase();
            if (capQuery.startsWith("SELECT")
                    && capQuery.indexOf("UPDATE") == -1) {
                throw new DBException(
                        "executeUpdate() can not execute SELECT queries. Use executeQuery() instead.");
            } else if (query.indexOf(';') != -1) {
                throw new DBException(
                        "executeUpdate() does not support semi-colons(;). Please issue individual DMLs separately.");
            }

            Session session = _getSession(host);
            int result = session.executeUpdate(query, values);
            if (this.isLogEnabled())
                Logging.logDBCall("Host=" + host.getName() + ", Query={"
                        + query + "}, values={" + Utils.dumpArray(values)
                        + "}, " + result + " row(s) updated.", false);

            return result;
        } catch (SeQuaLiteException e) {
            Logging.logDBCall("Host=" + host.getName() + ", Query={" + query
                    + "}, values={" + Utils.dumpArray(values) + "}", true);
            throw new DBException(e);
        }
    }

    private boolean isLogEnabled() {
        return ContextManager.getThreadContext().isDBLogEnabled();
    }

    public void loadHostConfig(Properties prop, DBHost hostInfo) {
        String jdbcContext = hostInfo.getName();
        prop.put(jdbcContext + ".jdbc.driver", hostInfo.getDriver());
        if (hostInfo.getUrl() == null)
            prop.put(jdbcContext + ".jdbc.url",
                    "jdbc:oracle:thin:@" + hostInfo.getHostAddress() + ":"
                            + hostInfo.getPort() + ":" + hostInfo.getSid());
        else
            prop.put(jdbcContext + ".jdbc.url", hostInfo.getUrl());
        prop.put(jdbcContext + ".jdbc.user", hostInfo.getUser());
        prop.put(jdbcContext + ".jdbc.password", hostInfo.getPassword());
        prop.put(jdbcContext + ".jdbc.autocommit", "false");
        prop.put(jdbcContext + ".pool.maxsize", "50");
    }

    /**
     * Rolls back to the DBHost for the current thread.
     *
     * @throws DBException
     */
    public void rollback() throws DBException {
        /*
         * if (threadLocalDBHostStack.get() == null ||
         * threadLocalDBHostStack.get().isEmpty()) throw new
         * DBException("You haven't executed a query on any DBHosts yet.");
         */
        if (threadLocalDBHostStack.get() != null
                && !threadLocalDBHostStack.get().isEmpty())
            rollback(threadLocalDBHostStack.get().peek());
    }
```




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