Code Inspection Document

Navid Heidari (798726) Hamidreza Hanafi (841408)

June 18, 2016

Contents

1 Classes and methods

1.1 Location

netty-socketio/src/main/java/com/corundumstudio/socketio/handler/EncoderHandler.java

1.2 Namespace

com.corundumstudio.socketio.handler

1.3 Class name

EncoderHandler

1.4 Analyzed methods

- **Method 1**: *EncoderHandler*(Configuration configuration, PacketEncoder encoder)
- **Method 2**: *readVersion*()
- **Method 3**: *write*(XHROptionsMessage msg, ChannelHandlerContext ctx, ChannelPromise promise)
- **Method 4**: *write*(XHRPostMessage msg, ChannelHandlerContext ctx, ChannelPromise promise)
- **Method 5**: *sendMessage*(HttpMessage msg, Channel channel, ByteBuf out, String type, ChannelPromise promise, HttpResponseStatus status)
- **Method 6**: *sendMessage*(HttpMessage msg, Channel channel, ByteBuf out, HttpResponse res, ChannelPromise promise)
- **Method 7**: *sendError*(HttpErrorMessage errorMsg, ChannelHandlerContext ctx, ChannelPromise promise)
- **Method 8**: *addOriginHeaders*(String origin, HttpResponse res)
- **Method 9**: *write*(ChannelHandlerContext ctx, Object msg, ChannelPromise promise)
- **Method 10**: *handleWebsocket*(final OutPacketMessage msg, ChannelHandlerContext ctx, ChannelPromise promise)
- **Method 11**: *handleHTTP*(OutPacketMessage msg, ChannelHandlerContext ctx, ChannelPromise promise)

2 Functional role of the class

There is no JavaDoc documention for this class and really we don't know the functional role for this class.

3 Issues found by applying the checklist

We use the following notation:

- ✓: the relative point in the checklist is satisfied by the method
- ✗: the relative point in the checklist is not satisfied and will follow the piece of code affected by the problem or a description of the problem

3.1 Naming Conventions

1. All class names, interface names, method names, class variables, method variables, and constants used should have meaningful names and do what the name suggests:
   - Method 1: ✓
   - Method 2: ✓
   - Method 3: ✓
   - Method 4: ✓
   - Method 5: ✓
   - Method 6: ✓
   - Method 7: ✓
   - Method 8: ✓
   - Method 9: ✓
   - Method 10: ✓
   - Method 11: ✓
2. If one-character variables are used, they are used only for temporary "throwaway" variables, such as those used in for loops.
   - Method 1: ✓
   - Method 2: ✓
   - Method 3: ✓
   - Method 4: ✓
   - Method 5: ✓
   - Method 6: ✓
   - Method 7: ✓
   - Method 8: ✓
   - Method 9: ✓
   - Method 10: ✓

- Method 11: ✓

3. Class names are nouns, in mixed case, with the first letter of each word in capitalized.
   - Class: ✓
4. Interface names should be capitalized like classes
   - No Interface
5. Method names should be verbs, with the first letter of each addition word capitalized.
   - Method 1: ✓
   - Method 2: ✓
   - Method 3: ✓
   - Method 4: ✓
   - Method 5: ✓
   - Method 6: ✓
   - Method 7: ✓
   - Method 8: ✓
   - Method 9: ✓
   - Method 10: ✓
   - Method 11: ✓
6. Class variables, also called attributes, are mixed case, but might begin with an underscore ('_') followed by a lowercase first letter. All the remaining words in the variable name have their first letter capitalized
   - Class: ✓
7. Constants are declared using all uppercase with words separated by an underscore
   - Class: ✓

3.2 Indention

8. Three or four spaces are used for indentation and done so consistently:
   - Method 1: ✓
   - Method 2: ✓
   - Method 3: ✓
   - Method 4: ✓
   - Method 5: ✓
   - Method 6: ✓
   - Method 7: ✓
   - Method 8: ✓
   - Method 9: ✓

- Method 10: ✓
- Method 11: ✓

9. No tabs are used to indent:

   - Method 1: ✓
   - Method 2: ✓
   - Method 3: ✓
   - Method 4: ✓
   - Method 5: ✓
   - Method 6: ✓
   - Method 7: ✓
   - Method 8: ✓
   - Method 9: ✓
   - Method 10: ✓
   - Method 11: ✓

3.3 Braces

10. Consistent bracing style is used, either the preferred Allman style (first brace goes underneath the opening block) or the Kernighan and Ritchie style (first brace is on the same line of the instruction that opens the new block) :

    - Method 1: ✓
    - Method 2: ✓
    - Method 3: ✓
    - Method 4: ✓
    - Method 5: ✓
    - Method 6: ✓
    - Method 7: ✓
    - Method 8: ✓
    - Method 9: ✓
    - Method 10: ✓
    - Method 11: ✓

11. All if, while, do-while, try-catch, and for statements that have only one statement to execute are surrounded by curly braces:

    - Method 1: ✓
    - Method 2: ✓
    - Method 3: ✓
    - Method 4: ✓
    - Method 5: ✓

- Method 6: ✓
- Method 7: ✓
- Method 8: ✓
- Method 9: ✓
- Method 10: ✓
- Method 11: ✓

3.4 File organization

12. Blank lines and optional comments are used to separate sections (beginning comments, package/import statements, class/interface declarations which include class variable/attributes declarations, constructors, and methods) :

    - Method 1: ✓
      Blank line is used but there is no optional comment.
    - Method 2: ✓
      Blank line is used but there is no optional comment.
    - Method 3: ✓
      Blank line is used but there is no optional comment.
    - Method 4: ✓
      Blank line is used but there is no optional comment.
    - Method 5: ✓
      Blank line is used but there is no optional comment.
    - Method 6: ✓
      Blank line is used but there is no optional comment.
    - Method 7: ✓
      Blank line is used but there is no optional comment.
    - Method 8: ✓
      Blank line is used but there is no optional comment.
    - Method 9: ✓
      Blank line is used but there is no optional comment.
    - Method 10: ✓
      Blank line is used but there is no optional comment.
    - Method 11: ✓
      Blank line is used but there is no optional comment.

13. Where practical, line length does not exceed 80 characters:

    - Class: ✗
      Often in the code, lines exceed 80 characters.

```
public static final AttributeKey<String> ORIGIN = AttributeKey.valueOf("
  origin");
```

**

```
public static final AttributeKey<String> USER_AGENT = AttributeKey.valueOf(
  "userAgent");
```

**

```
  public static final AttributeKey<Integer> JSONP_INDEX = AttributeKey.
    valueOf("jsonpIndex");
```

**

```
  public static final AttributeKey<Boolean> WRITE_ONCE = AttributeKey.valueOf
    ("writeOnce");
```

- Method 1: ✗
  Often in the code, lines exceed 80 characters.

```
  public EncoderHandler(Configuration configuration, PacketEncoder encoder)
    throws IOException {
```

- Method 2: ✗
  Often in the code, lines exceed 80 characters.

```
    Enumeration<URL> resources = getClass().getClassLoader().getResources("
    META-INF/MANIFEST.MF");
```

- Method 3: ✗
  Often in the code, lines exceed 80 characters.

```
  private void write(XHROptionsMessage msg, ChannelHandlerContext ctx,
    ChannelPromise promise) {
```

**

```
          .add(HttpHeaderNames.ACCESS_CONTROL_ALLOW_HEADERS, HttpHeaderNames.
    CONTENT_TYPE);
```

- Method 4: ✗
  Often in the code, lines exceed 80 characters.

```
  private void write(XHRPostMessage msg, ChannelHandlerContext ctx,
    ChannelPromise promise) {
```

**

```
    sendMessage(msg, ctx.channel(), out, "text/html", promise,
    HttpResponseStatus.OK);
```

- Method 5: ✗
  Often in the code, lines exceed 80 characters.

```
  private void sendMessage(HttpMessage msg, Channel channel, ByteBuf out,
    String type, ChannelPromise promise, HttpResponseStatus status) {
```

**

```
    if (userAgent != null && (userAgent.contains(";MSIE") || userAgent.
    contains("Trident/"))) {
```

- Method 6: ✗
  Often in the code, lines exceed 80 characters.

```
  private void sendMessage(HttpMessage msg, Channel channel, ByteBuf out,
    HttpResponse res, ChannelPromise promise) {
```

**

```
        log.trace("Out␣message:␣{}␣-␣sessionId:␣{}", out.toString(CharsetUtil
    .UTF_8), msg.getSessionId());
```

**

```
    channel.writeAndFlush(LastHttpContent.EMPTY_LAST_CONTENT, promise).
    addListener(ChannelFutureListener.CLOSE);
```

- Method 7: ✗
  Often in the code, lines exceed 80 characters.

```
  private void sendError(HttpErrorMessage errorMsg, ChannelHandlerContext ctx
    , ChannelPromise promise) throws IOException {
```

**

```
    sendMessage(errorMsg, ctx.channel(), encBuf, "application/json", promise,
     HttpResponseStatus.BAD_REQUEST);
```

- Method 8: ✗
  Often in the code, lines exceed 80 characters.

```
      res.headers().add(HttpHeaderNames.ACCESS_CONTROL_ALLOW_ORIGIN,
    configuration.getOrigin());
```

**

```
      res.headers().add(HttpHeaderNames.ACCESS_CONTROL_ALLOW_CREDENTIALS,
    Boolean.TRUE);
```

**

```
        res.headers().add(HttpHeaderNames.ACCESS_CONTROL_ALLOW_CREDENTIALS,
    Boolean.TRUE);
```

- Method 9: ✗
  Often in the code, lines exceed 80 characters.

```
  public void write(ChannelHandlerContext ctx, Object msg, ChannelPromise
    promise) throws Exception {
```

- Method 10: ✗
  Often in the code, lines exceed 80 characters.

```
  private void handleWebsocket(final OutPacketMessage msg,
    ChannelHandlerContext ctx, ChannelPromise promise) throws IOException {
```

**

```
        log.trace("Out␣message:␣{}␣sessionId:␣{}", out.toString(CharsetUtil.
    UTF_8), msg.getSessionId());
```

**

```
          log.trace("Out␣attachment:␣{}␣sessionId:␣{}", ByteBufUtil.hexDump(
    outBuf), msg.getSessionId());
```

- Method 11: ✗
  Often in the code, lines exceed 80 characters.

```
  private void handleHTTP(OutPacketMessage msg, ChannelHandlerContext ctx,
    ChannelPromise promise) throws IOException {

                                    **

      sendMessage(msg, channel, out, "application/octet-stream", promise,
    HttpResponseStatus.OK);
```

14. When line length must exceed 80 characters, it does NOT exceed 120 characters:
    - Method 1: ✓
    - Method 2: ✓
    - Method 3: ✓
    - Method 4: ✓
    - Method 5: ✓
    - Method 6: ✓
    - Method 7: ✓
    - Method 8: ✓
    - Method 9: ✓
    - Method 10: ✓
    - Method 11: ✓

3.5 Wrapping Lines

15. Line break occurs after a comma or an operator :
    - All Methods: ✗
      This never happens. Not even in the method declaration.
16. Higher-level breaks are used:
    - All Methods: ✗
      It does not use any breaks so this one has not happened.
17. A new statement is aligned with the beginning of the expression at the same level as the previous line:
    - Method 1: ✓
    - Method 2: ✓
    - Method 3: ✓
    - Method 4: ✓
    - Method 5: ✓
    - Method 6: ✓
    - Method 7: ✓
    - Method 8: ✓
    - Method 9: ✓
    - Method 10: ✓
    - Method 11: ✓

3.6 Comments

18. Comments are used to adequately explain what the class, interface, methods, and blocks of code are doing.

    - Class and All Methods: ✗
      There is no comment at all. Neither for Class nor Methods.
    - Blocks: ✗
      There is a few comment with a line of description. Just one or two.

19. Commented out code contains a reason for being commented out and a date it can be removed from the source file if determined it is no longer needed.

    - Method 1: ✓
    - Method 2: ✓
    - Method 3: ✓
    - Method 4: ✓
    - Method 5: ✓
    - Method 6: ✓
    - Method 7: ✓
    - Method 8: ✓
    - Method 9: ✓
    - Method 10: ✓
    - Method 11: ✓

3.7 Java Source Files

20. Each Java source file contains a single public class or interface.

    - Class: ✓

21. The public class is the first class or interface in the file.

    - Class: ✓

22. Check that the external program interfaces are implemented consistently with what is described in the javadoc.

    - Class: ✓

23. Check that the javadoc is complete

    - Method 1: ✗
      The Javadoc is not complete: it does not explain what this method is for and does not describe the kind and the role of the output of this method.
    - Method 2: ✗
      The Javadoc is not complete: it does not explain what this method is for and does not describe the kind and the role of the output of this method.

- Method 3: ✗
  The Javadoc is not complete: it does not explain what this method is for and does not describe the kind and the role of the output of this method.
- Method 4: ✗
  The Javadoc is not complete: it does not explain what this method is for and does not describe the kind and the role of the output of this method.
- Method 5: ✗
  The Javadoc is not complete: it does not explain what this method is for and does not describe the kind and the role of the output of this method.
- Method 6: ✗
  The Javadoc is not complete: it does not explain what this method is for and does not describe the kind and the role of the output of this method.
- Method 7: ✗
  The Javadoc is not complete: it does not explain what this method is for and does not describe the kind and the role of the output of this method.
- Method 8: ✗
  The Javadoc is not complete: it does not explain what this method is for and does not describe the kind and the role of the output of this method.
- Method 9: ✗
  The Javadoc is not complete: it does not explain what this method is for and does not describe the kind and the role of the output of this method.
- Method 10: ✗
  The Javadoc is not complete: it does not explain what this method is for and does not describe the kind and the role of the output of this method.
- Method 11: ✗
  The Javadoc is not complete: it does not explain what this method is for and does not describe the kind and the role of the output of this method.

3.8 Package import statements

24. If any package statements are needed, they should be the first noncomment statements. Import statements follow.

    - Class: ✓

3.9 Class and Interface Declarations

25. The class or interface declarations shall be in the following order :

    A. class/interface documentation comment

    B. class or interface statement

    C. class/interface implementation comment, if necessary

    D. class (static) variables

       a. first public class variables
       b. next protected class variables
       c. next package level (no access modifier)

d. last private class variables

E. instance variables
   a. first public instance variables
   b. next protected instance variables
   c. next package level (no access modifier)
   d. last private instance variables

F. constructors

G. methods

- Class: ✗
  A private static variable comes before public ones.

26. Methods are grouped by functionality rather than by scope or accessibility:
    - Class: ✓
27. Check that the code is free of duplicates, long methods, big classes, breaking encapsulation, as well as if coupling and cohesion are adequate:
    - Class: ✓

3.10 Initialization and Declarations

28. Check that variables and class members are of the correct type. Check that they have the right visibility (public/private/protected)
    - Method 1: ✓
    - Method 2: ✓
    - Method 3: ✓
    - Method 4: ✓
    - Method 5: ✓
    - Method 6: ✓
    - Method 7: ✓
    - Method 8: ✓
    - Method 9: ✓
    - Method 10: ✓
    - Method 11: ✓
29. Check that variables are declared in the proper scope
    - Method 1: ✓
    - Method 2: ✓
    - Method 3: ✓
    - Method 4: ✓
    - Method 5: ✓

- Method 6: ✓
- Method 7: ✓
- Method 8: ✓
- Method 9: ✓
- Method 10: ✓
- Method 11: ✓

30. Check that constructors are called when a new object is desired
    - Method 1: ✓
    - Method 2: ✓
    - Method 3: ✓
    - Method 4: ✓
    - Method 5: ✓
    - Method 6: ✓
    - Method 7: ✓
    - Method 8: ✓
    - Method 9: ✓
    - Method 10: ✓
    - Method 11: ✓
31. Check that all object references are initialized before use
    - Method 1: ✓
    - Method 2: ✓
    - Method 3: ✓
    - Method 4: ✓
    - Method 5: ✓
    - Method 6: ✓
    - Method 7: ✓
    - Method 8: ✓
    - Method 9: ✓
    - Method 10: ✓
    - Method 11: ✓
32. Variables are initialized where they are declared, unless dependent upon a computation
    - Method 1: ✓
    - Method 2: ✓
    - Method 3: ✓
    - Method 4: ✓
    - Method 5: ✓

- Method 6: ✓
- Method 7: ✓
- Method 8: ✓
- Method 9: ✓
- Method 10: ✓
- Method 11: ✓

33. Declarations appear at the beginning of blocks (A block is any code surrounded by curly braces '{' and '}'). The exception is a variable can be declared in a `for` loop

    - Method 1: ✓
    - Method 2: ✓
    - Method 3: ✗
      At line 118 and 121, a variable is declared after calling another method.

```
111   private void write(XHROptionsMessage msg, ChannelHandlerContext ctx,
        ChannelPromise promise) {
112     HttpResponse res = new DefaultHttpResponse(HTTP_1_1, HttpResponseStatus.
        OK);
113
114     res.headers().add(HttpHeaderNames.SET_COOKIE, "io=" + msg.getSessionId())
115           .add(HttpHeaderNames.CONNECTION, HttpHeaderValues.KEEP_ALIVE)
116           .add(HttpHeaderNames.ACCESS_CONTROL_ALLOW_HEADERS, HttpHeaderNames.
        CONTENT_TYPE);
117
118     String origin = ctx.channel().attr(ORIGIN).get();
119     addOriginHeaders(origin, res);
120
121     ByteBuf out = encoder.allocateBuffer(ctx.alloc());
122     sendMessage(msg, ctx.channel(), out, res, promise);
123   }
```

- Method 4: ✓
- Method 5: ✗
  At line 140 and 147, a variable is declared after calling another method.

```
131   private void sendMessage(HttpMessage msg, Channel channel, ByteBuf out,
        String type, ChannelPromise promise, HttpResponseStatus status) {
132     HttpResponse res = new DefaultHttpResponse(HTTP_1_1, status);
133
134     res.headers().add(HttpHeaderNames.CONTENT_TYPE, type)
135           .add(HttpHeaderNames.CONNECTION, HttpHeaderValues.KEEP_ALIVE);
136     if (msg.getSessionId() != null) {
137       res.headers().add(HttpHeaderNames.SET_COOKIE, "io=" + msg.getSessionId
        ());
138     }
139
140     String origin = channel.attr(ORIGIN).get();
141     addOriginHeaders(origin, res);
142
143     HttpUtil.setContentLength(res, out.readableBytes());
144
145     // prevent XSS warnings on IE
146     // https://github.com/LearnBoost/socket.io/pull/1333
147     String userAgent = channel.attr(EncoderHandler.USER_AGENT).get();
148     if (userAgent != null && (userAgent.contains(";MSIE") || userAgent.
        contains("Trident/"))) {
```

```
      res.headers().add("X-XSS-Protection", "0");
    }

    sendMessage(msg, channel, out, res, promise);
  }
```

- Method 6: ✓
- Method 7: ✓
- Method 8: ✓
- Method 9: ✓
- Method 5: ✗
  At line 234 and 237, a variable is declared after calling another method.

```
225   private void handleWebsocket(final OutPacketMessage msg,
        ChannelHandlerContext ctx, ChannelPromise promise) throws IOException {
226     while (true) {
227       Queue<Packet> queue = msg.getClientHead().getPacketsQueue(msg.
        getTransport());
228       Packet packet = queue.poll();
229       if (packet == null) {
230         promise.trySuccess();
231         break;
232       }
233
234       final ByteBuf out = encoder.allocateBuffer(ctx.alloc());
235       encoder.encodePacket(packet, out, ctx.alloc(), true);
236
237       WebSocketFrame res = new TextWebSocketFrame(out);
238       if (log.isTraceEnabled()) {
239         log.trace("Out␣message:␣{}␣sessionId:␣{}", out.toString(CharsetUtil.
        UTF_8), msg.getSessionId());
240       }
```

3.11 Method Calls

34. Check that parameters are presented in the correct order :
    - Method 1: ✓
    - Method 2: ✓
    - Method 3: ✓
    - Method 4: ✓
    - Method 5: ✓
    - Method 6: ✓
    - Method 7: ✓
    - Method 8: ✓
    - Method 9: ✓
    - Method 10: ✓
    - Method 11: ✓
35. Check that the correct method is being called, or should it be a different method with a similar name:

- Method 1: ✓
- Method 2: ✓
- Method 3: ✓
- Method 4: ✓
- Method 5: ✓
- Method 6: ✓
- Method 7: ✓
- Method 8: ✓
- Method 9: ✓
- Method 10: ✓
- Method 11: ✓

36. Check that method returned values are used properly:

    - Method 1: ✓
    - Method 2: ✓
    - Method 3: ✓
    - Method 4: ✓
    - Method 5: ✓
    - Method 6: ✓
    - Method 7: ✓
    - Method 8: ✓
    - Method 9: ✓
    - Method 10: ✓
    - Method 11: ✓

3.12 Arrays

37. Check that there are no off-by-one errors in array indexing (that is, all required array elements are correctly accessed through the index):

    - Method 1: ✓
    - Method 2: ✓
    - Method 3: ✓
    - Method 4: ✓
    - Method 5: ✓
    - Method 6: ✓
    - Method 7: ✓
    - Method 8: ✓
    - Method 9: ✓
    - Method 10: ✓

- Method 11: ✓

38. Check that all array (or other collection) indexes have been prevented from going out-of-bounds:
    - Method 1: ✓
    - Method 2: ✓
    - Method 3: ✓
    - Method 4: ✓
    - Method 5: ✓
    - Method 6: ✓
    - Method 7: ✓
    - Method 8: ✓
    - Method 9: ✓
    - Method 10: ✓
    - Method 11: ✓
39. Check that constructors are called when a new array item is desired:
    - Method 1: ✓
    - Method 2: ✓
    - Method 3: ✓
    - Method 4: ✓
    - Method 5: ✓
    - Method 6: ✓
    - Method 7: ✓
    - Method 8: ✓
    - Method 9: ✓
    - Method 10: ✓
    - Method 11: ✓

3.13 Object Comparisons

40. Check that all objects (including Strings) are compared with "equals" and not with "=="
    - Method 1: ✓
    - Method 2: ✓
    - Method 3: ✓
    - Method 4: ✓
    - Method 5: ✓
    - Method 6: ✓
    - Method 7: ✓
    - Method 8: ✓

- Method 9: ✗
  At line 210 and 213, used "==".

```
208       if (msg instanceof OutPacketMessage) {
209         OutPacketMessage m = (OutPacketMessage) msg;
210         if (m.getTransport() == Transport.WEBSOCKET) {
211           handleWebsocket((OutPacketMessage) msg, ctx, promise);
212         }
213         if (m.getTransport() == Transport.POLLING) {
214           handleHTTP((OutPacketMessage) msg, ctx, promise);
215         }
216       } else if (msg instanceof XHROptionsMessage) {
```

- Method 10: ✓
- Method 11: ✓

3.14 Output format

41. Check that displayed output is free of spelling and grammatical errors:

    - Method 1: ✓
    - Method 2: ✓
    - Method 3: ✓
    - Method 4: ✓
    - Method 5: ✓
    - Method 6: ✓
    - Method 7: ✓
    - Method 8: ✓
    - Method 9: ✓
    - Method 10: ✓
    - Method 11: ✓

42. Check that error messages are comprehensive and provide guidance as to how to correct the problem:

    - Method 1: ✓
    - Method 2: ✓
    - Method 3: ✓
    - Method 4: ✓
    - Method 5: ✓
    - Method 6: ✓
    - Method 7: ✓
    - Method 8: ✓
    - Method 9: ✓
    - Method 10: ✓
    - Method 11: ✓

43. Check that the output is formatted correctly in terms of line stepping and spacing:

- Method 1: ✓
- Method 2: ✓
- Method 3: ✓
- Method 4: ✓
- Method 5: ✓
- Method 6: ✓
- Method 7: ✓
- Method 8: ✓
- Method 9: ✓
- Method 10: ✓
- Method 11: ✓

3.15 Computation, Comparisons and Assignments

44. Check that the implementation avoids 'brutish programming':

- Method 1: ✓
- Method 2: ✓
- Method 3: ✓
- Method 4: ✓
- Method 5: ✓
- Method 6: ✓
- Method 7: ✓
- Method 8: ✓
- Method 9: ✓
- Method 10: ✓
- Method 11: ✓

45. Check order of computation/evaluation, operator precedence and parenthesizing:

- Method 1: ✓
- Method 2: ✓
- Method 3: ✓
- Method 4: ✓
- Method 5: ✓
- Method 6: ✓
- Method 7: ✓
- Method 8: ✓
- Method 9: ✓
- Method 10: ✓

- Method 11: ✓

46. Check the liberal use of parenthesis is used to avoid operator precedence problems:
    - Method 1: ✓
    - Method 2: ✓
    - Method 3: ✓
    - Method 4: ✓
    - Method 5: ✓
    - Method 6: ✓
    - Method 7: ✓
    - Method 8: ✓
    - Method 9: ✓
    - Method 10: ✓
    - Method 11: ✓
47. Check that all denominators of a division are prevented from being zero:
    - Method 1: ✓
    - Method 2: ✓
    - Method 3: ✓
    - Method 4: ✓
    - Method 5: ✓
    - Method 6: ✓
    - Method 7: ✓
    - Method 8: ✓
    - Method 9: ✓
    - Method 10: ✓
    - Method 11: ✓
48. Check that integer arithmetic, especially division, are used appropriately to avoid causing unexpected truncation/rounding:
    - Method 1: ✓
    - Method 2: ✓
    - Method 3: ✓
    - Method 4: ✓
    - Method 5: ✓
    - Method 6: ✓
    - Method 7: ✓
    - Method 8: ✓
    - Method 9: ✓

- Method 10: ✓
- Method 11: ✓

49. Check that the comparison and Boolean operators are correct:
    - Method 1: ✓
    - Method 2: ✓
    - Method 3: ✓
    - Method 4: ✓
    - Method 5: ✓
    - Method 6: ✓
    - Method 7: ✓
    - Method 8: ✓
    - Method 9: ✓
    - Method 10: ✓
    - Method 11: ✓
50. Check throw-catch expressions, and check that the error condition is actually legitimate:
    - Method 1: ✓
    - Method 2: ✓
    - Method 3: ✓
    - Method 4: ✓
    - Method 5: ✓
    - Method 6: ✓
    - Method 7: ✓
    - Method 8: ✓
    - Method 9: ✓
    - Method 10: ✓
    - Method 11: ✓
51. Check that the code is free of any implicit type conversions:
    - Method 1: ✓
    - Method 2: ✓
    - Method 3: ✓
    - Method 4: ✓
    - Method 5: ✓
    - Method 6: ✓
    - Method 7: ✓
    - Method 8: ✓
    - Method 9: ✓
    - Method 10: ✓
    - Method 11: ✓

3.16 Exceptions

52. Check that the relevant exceptions are caught
    - Method 1: ✓
    - Method 2: ✓
    - Method 3: ✓
    - Method 4: ✓
    - Method 5: ✓
    - Method 6: ✓
    - Method 7: ✓
    - Method 8: ✓
    - Method 9: ✓
    - Method 10: ✓
    - Method 11: ✓
53. Check that the appropriate action are taken for each catch block
    - Method 1: ✓
    - Method 2: ✓
    - Method 3: ✓
    - Method 4: ✓
    - Method 5: ✓
    - Method 6: ✓
    - Method 7: ✓
    - Method 8: ✓
    - Method 9: ✓
    - Method 10: ✓
    - Method 11: ✓

3.17 Flow of control

54. In a switch statement, check that all cases are addressed by break or return
    - Method 1: ✓
    - Method 2: ✓
    - Method 3: ✓
    - Method 4: ✓
    - Method 5: ✓
    - Method 6: ✓
    - Method 7: ✓
    - Method 8: ✓

- Method 9: ✓
- Method 10: ✓
- Method 11: ✓

55. Check that all switch statements have a default branch
    - Method 1: ✓
    - Method 2: ✓
    - Method 3: ✓
    - Method 4: ✓
    - Method 5: ✓
    - Method 6: ✓
    - Method 7: ✓
    - Method 8: ✓
    - Method 9: ✓
    - Method 10: ✓
    - Method 11: ✓
56. Check that all loops are correctly formed, with the appropriate initialization, increment and termination expressions
    - Method 1: ✓
    - Method 2: ✓
    - Method 3: ✓
    - Method 4: ✓
    - Method 5: ✓
    - Method 6: ✓
    - Method 7: ✓
    - Method 8: ✓
    - Method 9: ✓
    - Method 10: ✓
    - Method 11: ✓

3.18 Files

57. Check that all files are properly declared and opened
    - Method 1: ✓
    - Method 2: ✓
    - Method 3: ✓
    - Method 4: ✓
    - Method 5: ✓
    - Method 6: ✓

- Method 7: ✓
- Method 8: ✓
- Method 9: ✓
- Method 10: ✓
- Method 11: ✓

58. Check that all files are closed properly, even in the case of an error
    - Method 1: ✓
    - Method 2: ✓
    - Method 3: ✓
    - Method 4: ✓
    - Method 5: ✓
    - Method 6: ✓
    - Method 7: ✓
    - Method 8: ✓
    - Method 9: ✓
    - Method 10: ✓
    - Method 11: ✓
59. Check that EOF conditions are detected and handled correctly
    - Method 1: ✓
    - Method 2: ✓
    - Method 3: ✓
    - Method 4: ✓
    - Method 5: ✓
    - Method 6: ✓
    - Method 7: ✓
    - Method 8: ✓
    - Method 9: ✓
    - Method 10: ✓
    - Method 11: ✓
60. Check that all file exceptions are caught and dealt with accordingly
    - Method 1: ✓
    - Method 2: ✓
    - Method 3: ✓
    - Method 4: ✓
    - Method 5: ✓
    - Method 6: ✓

- Method 7: ✓
- Method 8: ✓
- Method 9: ✓
- Method 10: ✓
- Method 11: ✓

4 Appendix

4.1 Working hours

- Navid Heidari: 6 hours
- Hamidreza Hanafi: 6 hours

4.2 Methods Code

4.2.1 *EncoderHandler*

```java
public EncoderHandler(Configuration configuration, PacketEncoder encoder) throws
  IOException {
  this.encoder = encoder;
  this.configuration = configuration;

  if (configuration.isAddVersionHeader()) {
    readVersion();
  }
}
```

4.2.2 *readVersion*

```java
private void readVersion() throws IOException {
  Enumeration<URL> resources = getClass().getClassLoader().getResources("META-INF/
  MANIFEST.MF");
  while (resources.hasMoreElements()) {
    try {
      Manifest manifest = new Manifest(resources.nextElement().openStream());
      Attributes attrs = manifest.getMainAttributes();
      if (attrs == null) {
        continue;
      }
      String name = attrs.getValue("Bundle-Name");
      if (name != null && name.equals("netty-socketio")) {
        version = name + "/" + attrs.getValue("Bundle-Version");
        break;
      }
    } catch (IOException E) {
      // skip it
    }
  }
}
```

4.2.3 *write*

```java
private void write(XHROptionsMessage msg, ChannelHandlerContext ctx, ChannelPromise
  promise) {
  HttpResponse res = new DefaultHttpResponse(HTTP_1_1, HttpResponseStatus.OK);

  res.headers().add(HttpHeaderNames.SET_COOKIE, "io=" + msg.getSessionId())
        .add(HttpHeaderNames.CONNECTION, HttpHeaderValues.KEEP_ALIVE)
        .add(HttpHeaderNames.ACCESS_CONTROL_ALLOW_HEADERS, HttpHeaderNames.
  CONTENT_TYPE);

  String origin = ctx.channel().attr(ORIGIN).get();
  addOriginHeaders(origin, res);

  ByteBuf out = encoder.allocateBuffer(ctx.alloc());
  sendMessage(msg, ctx.channel(), out, res, promise);
}
```

4.2.4 *write*

```
private void write(XHRPostMessage msg, ChannelHandlerContext ctx, ChannelPromise
  promise) {
  ByteBuf out = encoder.allocateBuffer(ctx.alloc());
  out.writeBytes(OK);
  sendMessage(msg, ctx.channel(), out, "text/html", promise, HttpResponseStatus.OK);
}
```

4.2.5 *sendMessage*

```java
private void sendMessage(HttpMessage msg, Channel channel, ByteBuf out, String type,
    ChannelPromise promise, HttpResponseStatus status) {
  HttpResponse res = new DefaultHttpResponse(HTTP_1_1, status);

  res.headers().add(HttpHeaderNames.CONTENT_TYPE, type)
        .add(HttpHeaderNames.CONNECTION, HttpHeaderValues.KEEP_ALIVE);
  if (msg.getSessionId() != null) {
    res.headers().add(HttpHeaderNames.SET_COOKIE, "io=" + msg.getSessionId());
  }

  String origin = channel.attr(ORIGIN).get();
  addOriginHeaders(origin, res);

  HttpUtil.setContentLength(res, out.readableBytes());

  // prevent XSS warnings on IE
  // https://github.com/LearnBoost/socket.io/pull/1333
  String userAgent = channel.attr(EncoderHandler.USER_AGENT).get();
  if (userAgent != null && (userAgent.contains(";MSIE") || userAgent.contains("
    Trident/"))) {
    res.headers().add("X-XSS-Protection", "0");
  }

  sendMessage(msg, channel, out, res, promise);
}
```

4.2.6 *sendMessage*

```java
private void sendMessage(HttpMessage msg, Channel channel, ByteBuf out, HttpResponse res, ChannelPromise promise) {
  channel.write(res);

  if (log.isTraceEnabled()) {
    if (msg.getSessionId() != null) {
      log.trace("Out␣message:␣{}␣-␣sessionId:␣{}", out.toString(CharsetUtil.UTF_8), msg.getSessionId());
    } else {
      log.trace("Out␣message:␣{}", out.toString(CharsetUtil.UTF_8));
    }
  }

  if (out.isReadable()) {
    channel.write(new DefaultHttpContent(out));
  } else {
    out.release();
  }

  channel.writeAndFlush(LastHttpContent.EMPTY_LAST_CONTENT, promise).addListener(ChannelFutureListener.CLOSE);
}
```

4.2.7 *sendError*

```java
  private void sendError(HttpErrorMessage errorMsg, ChannelHandlerContext ctx,
    ChannelPromise promise) throws IOException {
    final ByteBuf encBuf = encoder.allocateBuffer(ctx.alloc());
    ByteBufOutputStream out = new ByteBufOutputStream(encBuf);
    encoder.getJsonSupport().writeValue(out, errorMsg.getData());

    sendMessage(errorMsg, ctx.channel(), encBuf, "application/json", promise,
    HttpResponseStatus.BAD_REQUEST);
  }
```

4.2.8 *addOriginHeaders*

```java
  private void addOriginHeaders(String origin, HttpResponse res) {
    if (version != null) {
      res.headers().add(HttpHeaderNames.SERVER, version);
    }

    if (configuration.getOrigin() != null) {
      res.headers().add(HttpHeaderNames.ACCESS_CONTROL_ALLOW_ORIGIN, configuration.
    getOrigin());
      res.headers().add(HttpHeaderNames.ACCESS_CONTROL_ALLOW_CREDENTIALS, Boolean.TRUE)
    ;
    } else {
      if (origin != null) {
        res.headers().add(HttpHeaderNames.ACCESS_CONTROL_ALLOW_ORIGIN, origin);
        res.headers().add(HttpHeaderNames.ACCESS_CONTROL_ALLOW_CREDENTIALS, Boolean.
    TRUE);
      } else {
        res.headers().add(HttpHeaderNames.ACCESS_CONTROL_ALLOW_ORIGIN, "*");
      }
    }
  }
```

4.2.9 *write*

```java
public void write(ChannelHandlerContext ctx, Object msg, ChannelPromise promise)
  throws Exception {
  if (!(msg instanceof HttpMessage)) {
    super.write(ctx, msg, promise);
    return;
  }

  if (msg instanceof OutPacketMessage) {
    OutPacketMessage m = (OutPacketMessage) msg;
    if (m.getTransport() == Transport.WEBSOCKET) {
      handleWebsocket((OutPacketMessage) msg, ctx, promise);
    }
    if (m.getTransport() == Transport.POLLING) {
      handleHTTP((OutPacketMessage) msg, ctx, promise);
    }
  } else if (msg instanceof XHROptionsMessage) {
    write((XHROptionsMessage) msg, ctx, promise);
  } else if (msg instanceof XHRPostMessage) {
    write((XHRPostMessage) msg, ctx, promise);
  } else if (msg instanceof HttpErrorMessage) {
    sendError((HttpErrorMessage) msg, ctx, promise);
  }
}
```

4.2.10 *handleWebsocket*

```java
private void handleWebsocket(final OutPacketMessage msg, ChannelHandlerContext ctx, ChannelPromise promise) throws IOException {
  while (true) {
    Queue<Packet> queue = msg.getClientHead().getPacketsQueue(msg.getTransport());
    Packet packet = queue.poll();
    if (packet == null) {
      promise.trySuccess();
      break;
    }

    final ByteBuf out = encoder.allocateBuffer(ctx.alloc());
    encoder.encodePacket(packet, out, ctx.alloc(), true);

    WebSocketFrame res = new TextWebSocketFrame(out);
    if (log.isTraceEnabled()) {
      log.trace("Out␣message:␣{}␣sessionId:␣{}", out.toString(CharsetUtil.UTF_8), msg.getSessionId());
    }

    if (out.isReadable()) {
      if (!promise.isDone()) {
        ctx.channel().writeAndFlush(res, promise);
      } else {
        ctx.channel().writeAndFlush(res);
      }
    } else {
      promise.trySuccess();
      out.release();
    }

    for (ByteBuf buf : packet.getAttachments()) {
      ByteBuf outBuf = encoder.allocateBuffer(ctx.alloc());
      outBuf.writeByte(4);
      outBuf.writeBytes(buf);
      if (log.isTraceEnabled()) {
        log.trace("Out␣attachment:␣{}␣sessionId:␣{}", ByteBufUtil.hexDump(outBuf), msg.getSessionId());
      }
      ctx.channel().writeAndFlush(new BinaryWebSocketFrame(outBuf));
    }
  }
}
```

4.2.11 *handleHTTP*

```java
  private void handleHTTP(OutPacketMessage msg, ChannelHandlerContext ctx,
    ChannelPromise promise) throws IOException {
    Channel channel = ctx.channel();
    Attribute<Boolean> attr = channel.attr(WRITE_ONCE);

    Queue<Packet> queue = msg.getClientHead().getPacketsQueue(msg.getTransport());

    if (!channel.isActive() || queue.isEmpty() || !attr.compareAndSet(null, true)) {
      promise.trySuccess();
      return;
    }

    ByteBuf out = encoder.allocateBuffer(ctx.alloc());
    Boolean b64 = ctx.channel().attr(EncoderHandler.B64).get();
    if (b64 != null && b64) {
      Integer jsonpIndex = ctx.channel().attr(EncoderHandler.JSONP_INDEX).get();
      encoder.encodeJsonP(jsonpIndex, queue, out, ctx.alloc(), 50);
      String type = "application/javascript";
      if (jsonpIndex == null) {
        type = "text/plain";
      }
      sendMessage(msg, channel, out, type, promise, HttpResponseStatus.OK);
    } else {
      encoder.encodePackets(queue, out, ctx.alloc(), 50);
      sendMessage(msg, channel, out, "application/octet-stream", promise,
    HttpResponseStatus.OK);
    }
  }

}
```