HTTPMessageTypeTest.java

HTTPMessage::determineMessageType(String firstLine)

Valid Request Line: Ensure it returns an HTTPRequest when a valid request line is provided.
Valid Response Line: Ensure it returns an HTTPResponse when a valid status line is provided.
Invalid First Line: Ensure it throws IllegalArgumentException when the line does not adhere to the expected formats.
Empty First Line: Ensure it handles empty input strings and throws an appropriate exception.
Null First Line: Ensure it handles null inputs and throws an appropriate exception.
Malformed First Line: Test various malformed lines to ensure robust error handling.

HTTPMessageHeaderTest.java

HTTPMessage::setHeader(String headerName, String headerValue)

Verify each supported header type: ("Connection", "Content-Type", "Content-Length", etc.)
Valid Header: Ensure it sets the header correctly.
Invalid Header Value: Ensure it throws an IllegalArgumentException or logs warnings as per logic.
Null or Empty Header Name/Value: Ensure it handles null or empty inputs appropriately.
Unsupported Header: Ensure that it logs a warning for unsupported header names.
Negative or Non-integer Content-Length: Ensure it throws an IllegalArgumentException.

HTTPMessage::getHeader(String headerName) & HTTPMessage::getHeaders()

Existing Header: Ensure it retrieves the correct header value for an existing header.
Non-Existing Header: Ensure it returns null or behaves as per design for a non-existing header.
All Headers: For getHeaders(), ensure it returns all headers in the expected format.
No Headers: Ensure getHeaders() handles cases where no headers have been set.

HTTPMessageConnectionTest.java

HTTPMessage::shouldKeepConnectionAlive()

HTTP/1.0 Default: Ensure it returns false (or true only if "Connection: keep-alive" is present).
HTTP/1.1 Default: Ensure it returns true (or false if "Connection: close" is present).

Unspecified Protocol: Ensure it handles cases with an unspecified/unknown protocol appropriately.

HTTPMessageBodyTest.java

HTTPMessage::setBody(String body) & HTTPMessage::getBody()

Regular Body Content: Ensure it sets and gets the body content accurately.
Empty Body: Ensure it handles an empty body appropriately.
Null Body: Ensure it handles a null body as intended.
Large Body: Optionally, test with a large body to ensure that there are no issues with significant content sizes.

HTTPRequestTest.java

HTTPRequest::getRequestMethod()

Valid Request Method: Ensure it returns the correct HTTP method when it is set correctly (GET, POST, etc.).
Invalid Request Method: Ensure that if an invalid method is set, the class should handle it properly, possibly by throwing an exception or logging a warning.

HTTPRequest::getURI()

Valid URI: Ensure it returns the correct URI when a valid URI is set.
Invalid URI: Test with an invalid URI (like an empty string or a string with spaces) and verify that the class handles it properly.

HTTPRequest::getRequestLine()

Valid Components: Ensure it concatenates method, URI, and protocol version correctly when all are valid.
Missing Components: Ensure it handles scenarios where one or more components (method, URI, or protocol version) are missing or invalid.

HTTPRequest::toString()

All Valid Components: Ensure the string representation is accurate when all parts (request line, headers, body) are valid.
Partial/Invalid Components: Check the string output when some parts are missing or invalid to verify its handling and representing state accurately.

HTTPResponseTest.java

HTTPResponse::getStatusCode()

Valid Status Code: Ensure it returns the correct status code when a valid one is set.

Invalid Status Code: Ensure it manages invalid inputs like non-numeric, negative, or non-3-digit codes appropriately, maybe by rejecting them or logging warnings.

HTTPResponse::getResponseLine()

Valid Components: Ensure it concatenates the protocol version and status code correctly when valid.
Missing Components: Ensure it handles scenarios where one or more components (protocol version or status code) are missing or invalid.

HTTPResponse::toString()

All Valid Components: Ensure the string representation is accurate when all parts (status line, headers, body) are valid.
Partial/Invalid Components: Check the string output when some parts are missing or invalid to verify its handling and representing state accurately.