GUI:

The application has a menu bar with several options:

- "Start": starts the proxy server and displays a message on the GUI indicating that the server is running.
- "Stop": stops the proxy server and displays a message on the GUI indicating that the server has stopped.
- "Report": generates and displays a report of some kind (the implementation of this feature is not shown in the code provided).
- "Add host to filter": allows the user to add a host to a list of filtered addresses.
- "Display current filtered hosts": displays the list of filtered addresses on the GUI.
- "Exit": closes the application.

The application also implements the `Runnable` interface, which means it has a `run()` method that is executed when the application starts. However, the implementation of the `run()` method is not shown in the code provided.

Server Proxy:

The `ServerProxy` class appears to be a utility class that facilitates the communication between a client and a server through a proxy server. It has an input stream (`proxyToClientInput_Stream`) and an output stream (`proxyToServerOutput_Stream`) as fields and implements the `Runnable` interface. The input stream is used to read data from the client and the output stream is used to send data to the server.

The `ServerProxy` class has a `run()` method that reads data from the client input stream, byte by byte, and sends it directly to the server output stream. If the client input stream is no longer available, the data is flushed to the server output stream. The `run()` method is executed in a new thread when an instance of the `ServerProxy` class is created.

Utility Classes:

The `Response` class represents an HTTP response, which consists of a status code, a reason phrase, and a MIME header. The MIME header is represented by an instance of the `MimeHeader` class, which is a subclass of `HashMap<String, String>`. The `MimeHeader` class has a constructor that takes a string as an argument and parses the string to create key-value pairs that represent the MIME header fields. The `MimeHeader` class also has a `toString()` method that converts the MIME header fields back into a string representation.

The `MimeHeader` class has a `put()` method that adds a key-value pair to the MIME header, and a `toString()` method that converts the MIME header fields into a string representation. The `toString()` method iterates through the keys in the MIME header and appends each key-value pair to a string, separated by a colon and a space. The `toString()` method is used to convert the MIME header into a string representation that can be included in an HTTP response.

Server Handler:

The `ServerHandler` class appears to be a utility class that handles requests from clients to a server through a proxy server. It has a `DataInputStream` (`inFromClient`) and a `DataOutputStream` (`outToClient`) as fields, as well as a `host`, `path`, and `thread` field. The `ServerHandler` class implements the `Runnable` interface and has a `run()` method that is executed in a new thread when an instance of the `ServerHandler` class is created.

The `ServerHandler` class has several methods that facilitate the handling of client requests:

- `getRequestHeader(DataInputStream in)`: reads the request header from the client input stream and returns it as a string. It reads data from the input stream until it encounters a blank line, indicating the end of the request header.
- `createErrorPage(int code, String msg, String address)`: generates an HTML error page with the given status code, message, and address.
- `handleHeadHeader(MimeHeader requestMimeHeader, String request)`: handles an HTTP HEAD request by sending the response header to the client output stream. It also logs the request in a file.
- `handleGetHeader(MimeHeader requestMimeHeader, String request)`: handles an HTTP GET request by sending the response header and body to the client output stream. It also logs the request in a file.
- `handlePostHeader(MimeHeader requestMimeHeader, String request)`: handles an HTTP POST request by sending the response header and body to the client output stream. It also logs the request in a file.

The `run()` method of the `ServerHandler` class reads the request header from the client input stream, parses the header to extract the host and path, and then sends a request to the server specified in the host field. It then reads the response from the server and sends it back to the client.

If the request is an HTTP HEAD request, the `handleHeadHeader` method is called. If the request is an HTTP GET request, the `handleGetHeader` method is called. If the request is an HTTP POST request, the `handlePostHeader` method is called.

The `handleHeadHeader`, `handleGetHeader`, and `handlePostHeader` methods all follow a similar structure. They create a connection to the server specified in the host field, send the request to the server, read the response from the server, and then send the response back to the client. They also log the request in a file.

The `ServerProxy` class appears to be a utility class that facilitates communication between the proxy server and the server specified in the host field. It has an `InputStream` (`proxyToClientInput_Stream`) and an `OutputStream` (`proxyToServerOutput_Stream`) as fields. The `ServerProxy` class also implements the `Runnable` interface and has a `run()` method that is executed in a new thread when an instance of the `ServerProxy` class is created.

The `run()` method of the `ServerProxy` class reads data from the `proxyToClientInput_Stream` and writes it to the `proxyToServerOutput_Stream`. It continues to do this until the `proxyToClientInput_Stream` has no more data to read. The `getReadings` method is called to read data from the `proxyToClientInput_Stream` and write it to the `proxyToServerOutput_Stream`.

The `Response` class appears to be a utility class that represents an HTTP response. It has three fields: an `int` (`statusCode`) representing the status code of the response, a `String` (`reasonPhrase`) representing the reason phrase of the response, and a `MimeHeader` (`mh`) representing the MIME header of the response. The `Response` class has a `toString()` method that returns a string representation of the response.

The `MimeHeader` class appears to be a utility class that represents a MIME header. It extends the `HashMap` class and has a default constructor and a constructor that takes a `String` as an argument. The `MimeHeader` class has a `toString()` method that returns a string representation of the MIME header. It also has a `makeMimeHeader(String contentType, int length)`