ZHEN ZHANG Alex

22103518D

**Summary**

The server program presented across the various code snippets is essentially a simplified HTTP server implemented in Java. The server leverages Java's NIO (Non-blocking I/O) features, threading, and enums to handle HTTP requests efficiently. Below is a comprehensive summary of its design and implementation:

**Design Overview**

**Non-blocking I/O Model:**

The server utilizes Java's NIO capabilities, specifically ServerSocketChannel and Selector, which enable handling multiple client connections on a single thread. The ServerSocketChannel listens for incoming connections in a non-blocking manner, and the Selector is used to manage multiple channels, checking for readiness to perform operations like accept or read without blocking.

**Multithreaded Request Processing:**

Once a connection is established and ready to be read, the actual processing of HTTP requests is offloaded to a thread pool. This is managed by an ExecutorService, which facilitates handling multiple requests in parallel, thus improving the server's scalability.

**HTTP Status Code Handling:**

An enum StatusCode is used to clearly define various HTTP status codes and associated messages. This provides a robust, error-free mechanism for managing response statuses, making the code more readable and maintainable.

**Implementation Details**

**Initialization:**

The server is instantiated with a specific port and a fixed size for the thread pool. During initialization, the ServerSocketChannel is configured to non-blocking mode and registered with a Selector to monitor OP\_ACCEPT events, indicating that new connections can be accepted.

**Connection Acceptance and Registration:**

When the Selector detects an OP\_ACCEPT event on the server socket channel, the server accepts the client connection. The new SocketChannel for the client is also set to non-blocking mode and registered with the Selector to monitor OP\_READ events.

**Request Handling:**

When a Selector detects an OP\_READ event, indicating that data is ready to be read from a channel, the channel is passed to a worker thread from the thread pool. This thread is responsible for reading the data, processing the HTTP request, and sending back an appropriate response. The specifics of request parsing and response formation are abstracted out and presumably handled by another component like HttpHandler, which was referred to but not detailed in the provided snippets.

**Server Shutdown:**

The server can be stopped, which involves closing the server socket channel and shutting down the thread pool. However, actual invocation of the stop method is not reachable in the provided main method due to an infinite loop in the start method. This would require additional logic to handle graceful shutdown, such as listening for a shutdown command or integrating signal handling.

**HTTP Status Management:**

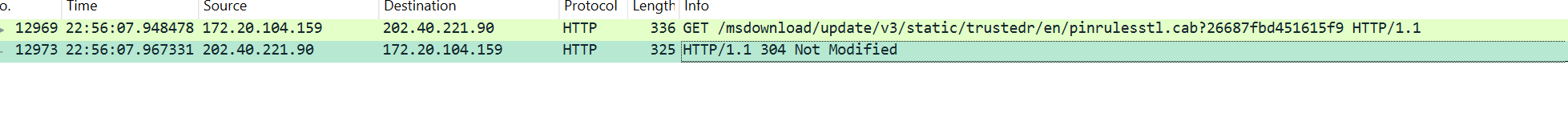
The StatusCode enum is used across the server to manage HTTP response statuses in a type-safe manner. This avoids magic numbers and strings throughout the code, reducing potential errors and improving clarity.

**Demonstration**

**404 Not found**

****

**304 Not Modified**

****

**400 Bad Request**

****

**200 OK**



**Log File**

/172.20.104.159:50373, 1713706261713, /projecttttt/wawwa.txt, null

/172.20.104.159:50373, 1713706261739, /favicon.ico, null

/172.20.104.159:50375, 1713706264467, /projecttttt/wawwa.txt, null

/172.20.104.159:50375, 1713706264488, /favicon.ico, null

/172.20.104.159:50376, 1713706266188, /projecttttt/wawwa.txt, null

/172.20.104.159:50376, 1713706266221, /favicon.ico, null

/172.20.104.159:50463, 1713706399589, /projecttttt/MONKEY.png, null

/172.20.104.159:50464, 1713706424046, /projecttttt/MONKEY.jpg, image/\*

A screenshot of a computer code

Description automatically generated