```java
import io.opentelemetry.api.GlobalOpenTelemetry;
import io.opentelemetry.api.trace.Span;
import io.opentelemetry.api.trace.Tracer;
import io.opentelemetry.api.trace.TracerProvider;
import io.opentelemetry.context.Scope;

import java.io.*;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.UUID;

public class server {
    public static void main(String[] args) {
        System.setProperty("otel.tracer.provider", "io.opentelemetry.api.trace.propagation.B3Propagator$Factory");

        try {
            // Create a server socket listening on port 8080
            ServerSocket serverSocket = new ServerSocket(8080);
            System.out.println("Server is listening on port 8080...");

            while (true) {
                // Wait for a client to connect
                Socket clientSocket = serverSocket.accept();
                System.out.println("Client connected.");

                // Start a new span for the server operation
                TracerProvider tracerProvider = GlobalOpenTelemetry.getTracerProvider();
                Tracer tracer = tracerProvider.get("server-tracer");

                // Start a span
                Span span = tracer.spanBuilder("
```

```
31 server-operation").startSpan();
32
33                 // Create a scope to manage the span'
   s lifecycle
34                 try (Scope scope = span.makeCurrent
   ()) {
35                     // Process the client's request
36                     processClientRequest(clientSocket
   );
37                 } finally {
38                     // End the span when the
   operation is complete
39                     span.end();
40                 }
41
42                 // Close the client socket
43                 clientSocket.close();
44             }
45         } catch (IOException e) {
46             e.printStackTrace();
47         }
48     }
49
50     private static void processClientRequest(Socket
   clientSocket) throws IOException {
51         // Create input stream for communication with
    the client
52         BufferedReader in = new BufferedReader(new
   InputStreamReader(clientSocket.getInputStream()));
53
54         // Read the number of files to expect
55         int numFiles;
56         try {
57             numFiles = Integer.parseInt(in.readLine
   ());
58         } catch (NumberFormatException e) {
59             System.err.println("Error reading the
   number of files from the client.");
60             return;
61         }
62         System.out.println("Expecting " + numFiles +
```

```
62 " files from the client.");
63
64         // Receive files from the client
65         for (int i = 0; i < numFiles; i++) {
66             // Generate a random file name
67             String fileName = UUID.randomUUID().
   toString() + ".txt";
68
69             // Read the file content from the client
70             StringBuilder fileContent = new
   StringBuilder();
71             String line;
72             while ((line = in.readLine()) != null
    && !line.equals("END_OF_FILE")) {
73                 fileContent.append(line).append("\n"
   );
74             }
75
76             // Save the file content to a file
77             try (FileOutputStream fileOutputStream
    = new FileOutputStream("received_" + fileName)) {
78                 fileOutputStream.write(fileContent.
   toString().getBytes());
79                 System.out.println("File content
   saved to received_" + fileName);
80             } catch (IOException e) {
81                 System.err.println("Error saving
   file content to received_" + fileName);
82                 e.printStackTrace();
83             }
84         }
85
86         // Close the input stream
87         in.close();
88     }
89 }
90
```