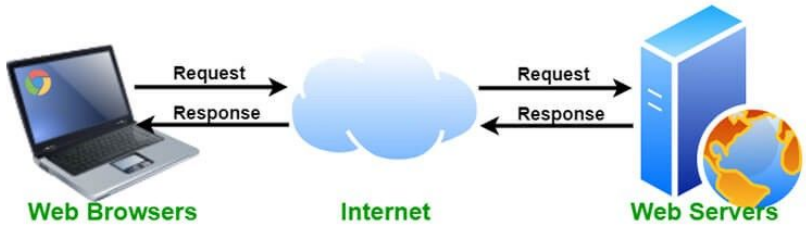
SWEN 262
Engineering of Software Subsystems

Singleton Pattern

Jiffy Web Server

1. A client may connect to the Jiffy Web Server (JWS).
   a. *Clients issue requests using HTTP.*
   b. *JWS responds to each request using HTTP.*
2. JWS must log information about each request, including:
   a. *Time/date stamp*
   b. *Source IP address*
   c. *HTTP request method (GET, POST, etc.)*
   d. *HTTP headers*
   e. *The HTTP response code and accompanying message (200 OK, 404 NOT FOUND, etc.).*
3. Multiple, concurrent clients may connect simultaneously.
   a. *JWS must respond to each request as quickly as possible.*

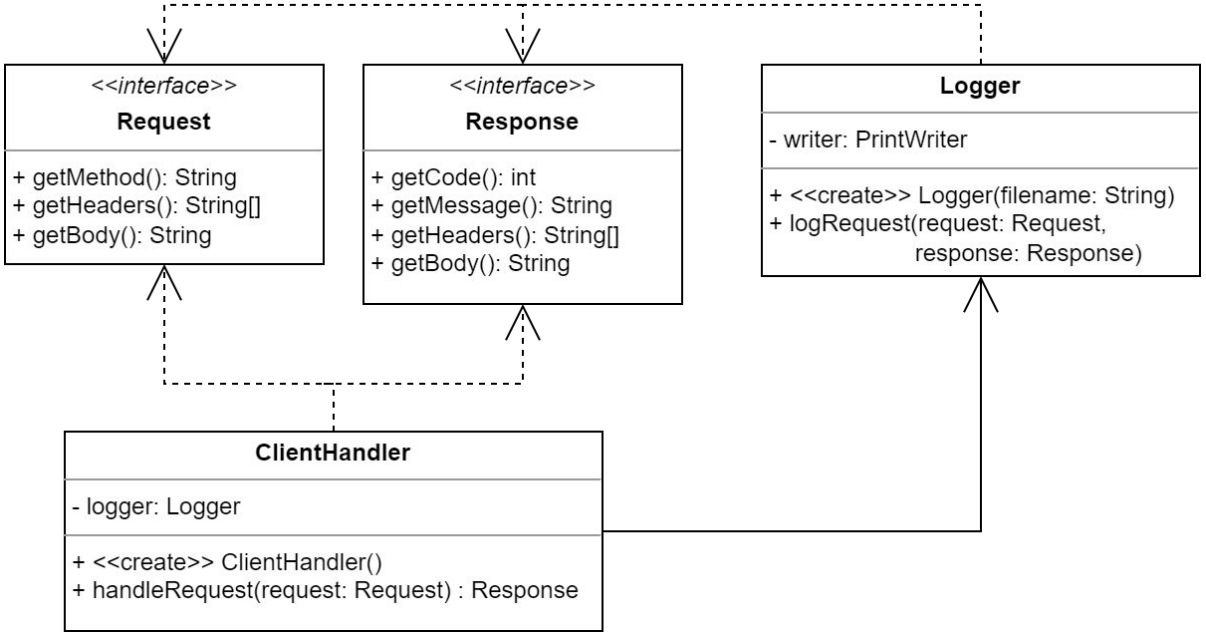


Every web server must support multiple, concurrent clients without blocking - handling requests from one client should not impact the performance of others.

This means that the server must be multi-threaded (or multi-process or multi-processor on a network for scaling), so that it can communicate with potentially many clients at the same time.

The web server must also log information about each request from any client.

Everyone Gets a Logger!



We will need a class that handles the request/response from each client - let's call it `ClientHandler`.

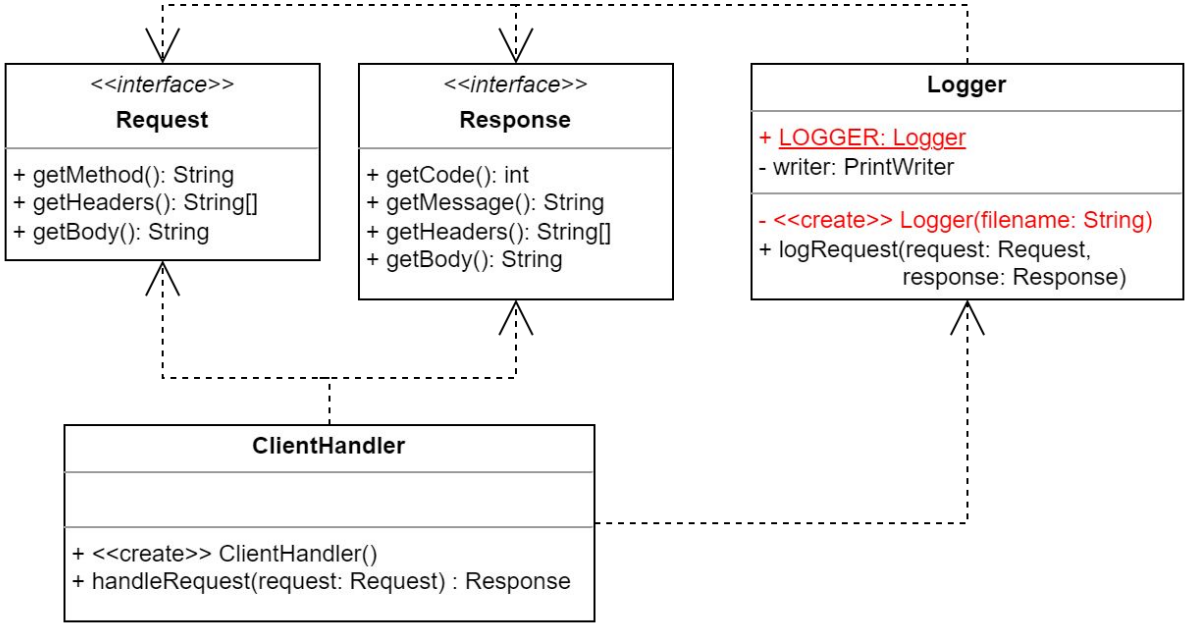
Each client handler will run in a separate thread so that we can handle multiple, concurrent clients at the same time.

One possible solution is that each client handler will create its own logger to log its requests all to the same place (file, database, etc.).

Q: What are the possible drawbacks of this approach?

A: What if two or more client handlers try to write to the log at the same time?

A Global Variable



Having many different threads all trying to write to the same log file at the same time will cause all kinds of problems.

What if we made a global variable in the `Logger` class that any client handler could access?

Q: What are the drawbacks of this solution?

A: When does the global get created? What if something goes wrong? How does it know to which file it should write? Aren't global variables kind of bad?

A Singleton Logger

```
public class Logger {
    public static final String LOGGER_FILENAME = "server.log";
    private static Logger INSTANCE;

    private final PrintWriter writer;

    private Logger() throws IOException {
        FileWriter fw = new FileWriter(LOGGER_FILENAME);
        writer = new PrintWriter(fw);
    }

    public static synchronized Logger instance() throws IOException {
        if(INSTANCE == null) {
            INSTANCE = new Logger();
        }
        return INSTANCE;
    }
}
```

A Singleton Logger

```
public class Logger {
    public static final String LOGGER_FILENAME = "server.log";
    private static Logger INSTANCE;

    private final PrintWriter writer;

    private Logger() throws IOException {
        FileWriter fw = new FileWriter(LOGGER_FILENAME);
        writer = new PrintWriter(fw);
    }

    public static synchronized Logger instance() throws IOException {
        if(INSTANCE == null) {
            INSTANCE = new Logger();
        }
        return INSTANCE;
    }
}
```

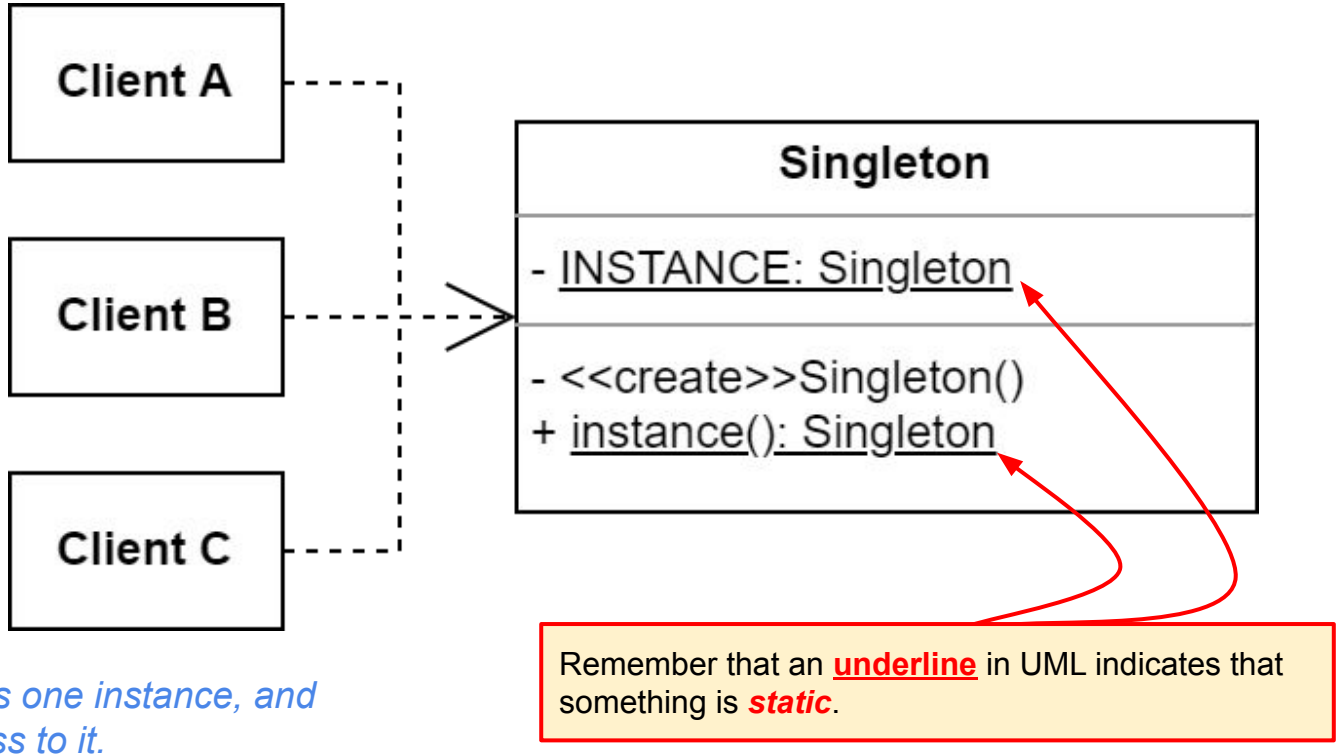
We make the `Logger` into a ***singleton*** starting with a ***private, global variable*** for the ***one*** `Logger` that everyone will use.

We also make sure that the ***constructor is private*** so that no other class can make a Logger.

Next, we provide a ***globally accessible method*** that creates a new instance of the `Logger` only if one does not already exist.

This is referred to as ***lazy initialization*** - the logger is only created at the moment that it is needed, and then reused later.

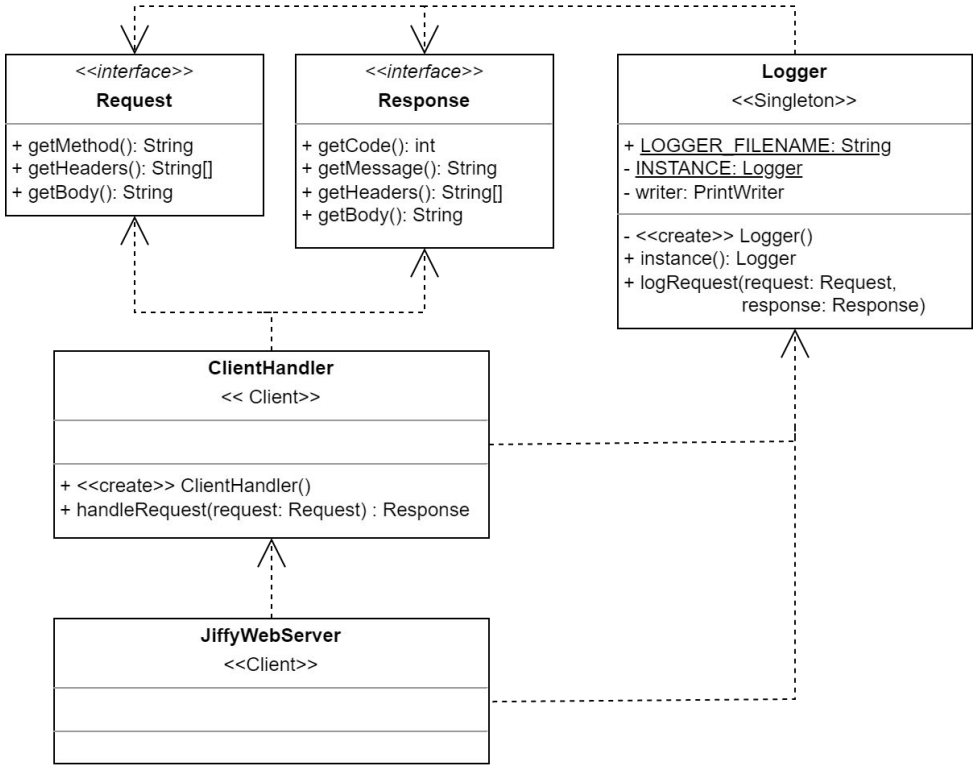
GoF Singleton Structure



Intent: Ensure a class only has one instance, and provide a global point of access to it.

(Creational)

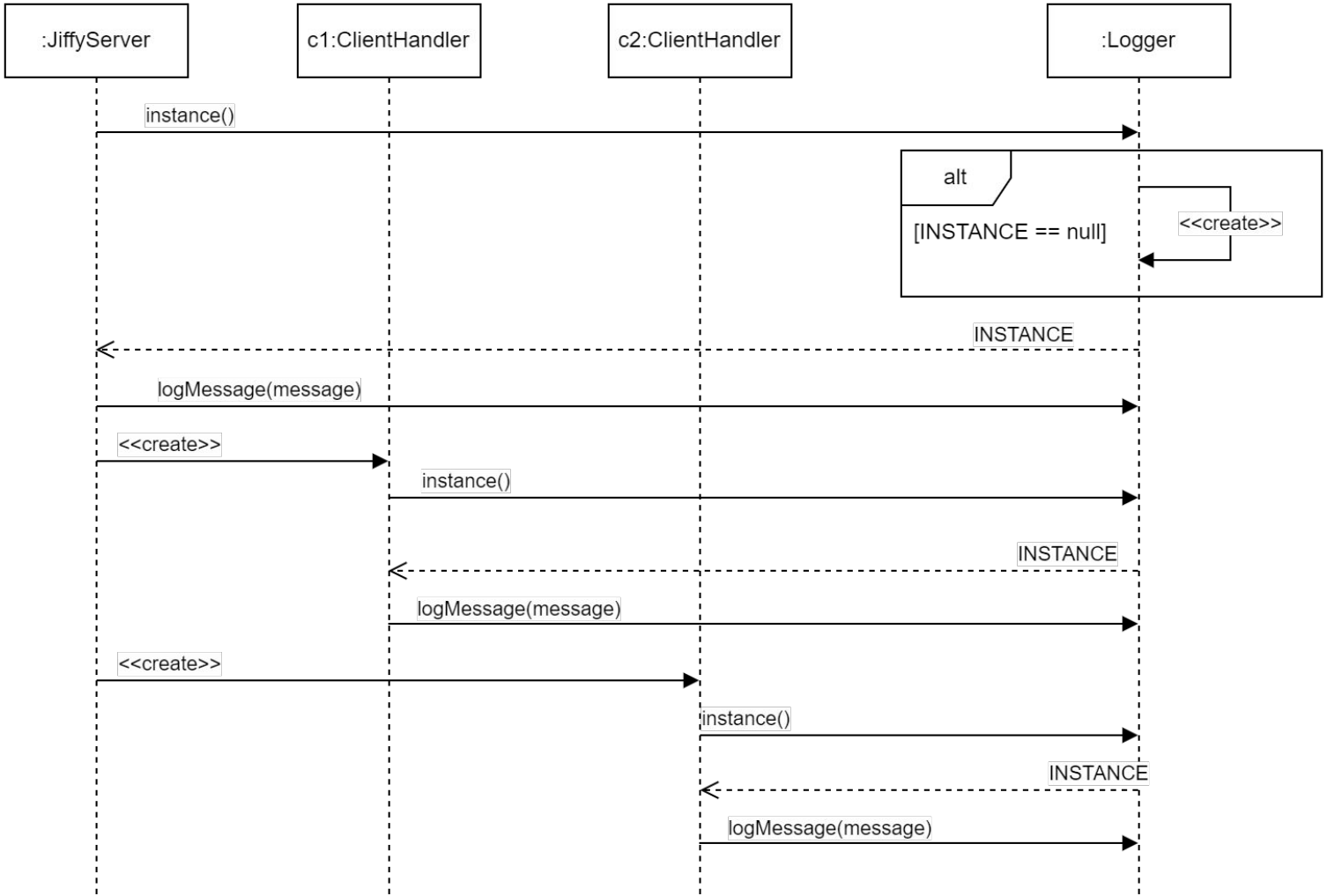
Jiffy Web Server System Design



As usual, each class has a context specific name...

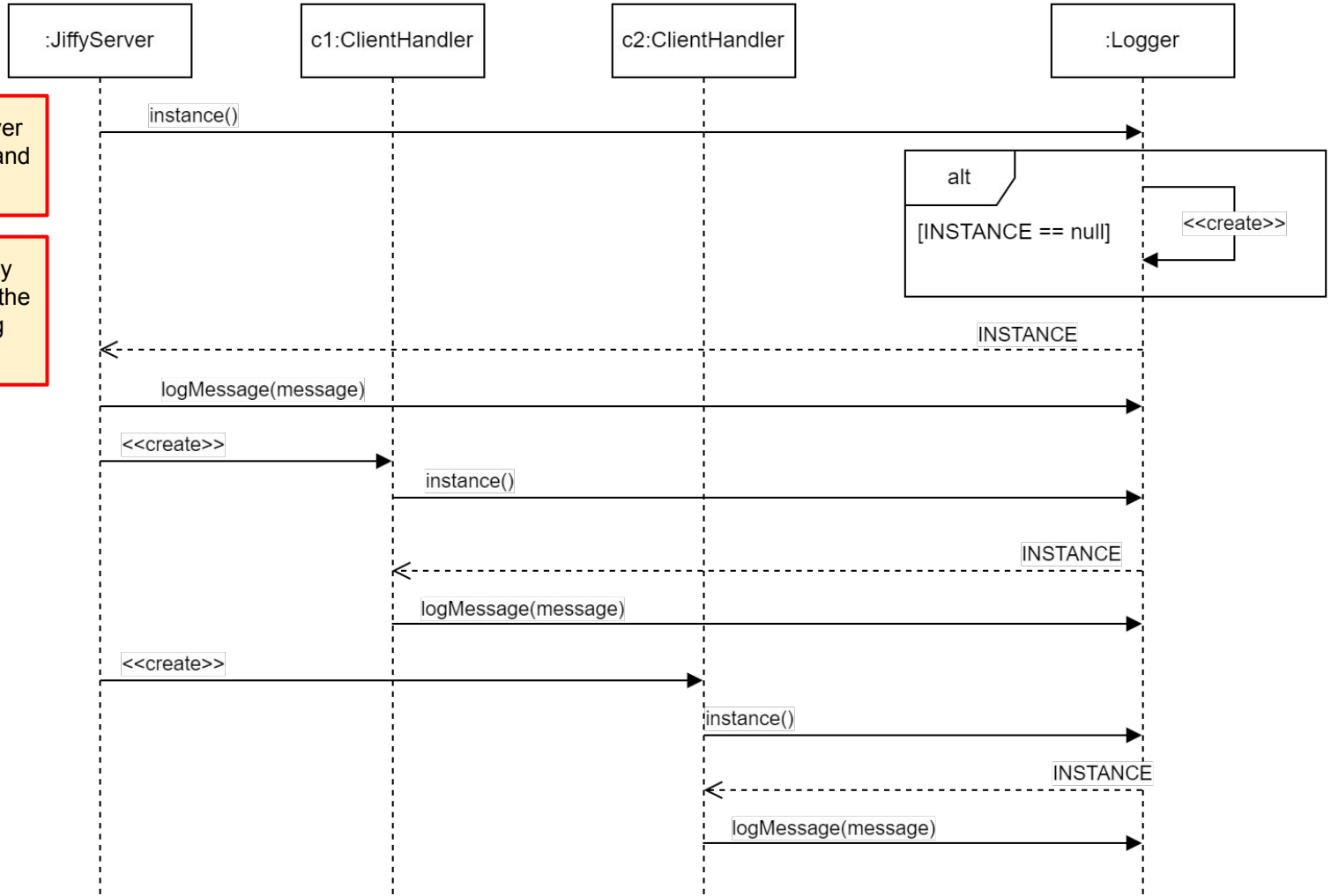
...but its role in the pattern is indicated in << *guillemets* >>.

Note that there may be many client handlers active at the same time.


:JiffyServer
c1:ClientHandler
c2:ClientHandler
:Logger
instance()
alt
[INSTANCE == null]
<<create>>
INSTANCE
logMessage(message)
<<create>>
instance()
INSTANCE
logMessage(message)
<<create>>
instance()
INSTANCE
logMessage(message)

This diagram shows the server getting the singleton logger and using it to log a message.

Later, as clients connect, they use the static method to get the same logger and use it to log messages.



GoF Pattern Card

Name: *Jiffy Web Server Logger*		GoF Pattern: *Singleton*
Participants		
Class	Role in Pattern	Participant's Contribution in the context of the application
Logger	*Singleton*	*The single, globally accessible logger in the system. It is responsible for logging messages from any client in a threadsafe way to the same file, database, etc.*
JiffyWebServer	*Client*	*The main Jiffy Web Server class. It creates client handler for each client request that is received by the server. It also logs messages related to server activity using the logger.*
ClientHandler	*Client*	*Handles requests from a single client. It uses the logger to log the status of each request/response including information about the HTTP methods, response code, error messages, and HTTP header.*
Deviations from the standard pattern: *None.*		
Requirements being covered: *2. JWS logs information about each request.*		

Indicators for Singleton

1. You must guarantee that there is ***at most*** one instance of a class.
2. The class is needed ***globally***; it is difficult to assign ownership/responsibility for creating the singleton to one other class in the system.
3. ***Maybe***: you need lazy initialization (you want to defer making the singleton until it is actually needed).

Singleton

There are several *consequences* to implementing the singleton pattern:

- *Ensures that exactly one instance of a class is created.*
- *Provides global access to the instance.*
- *Uses lazy initialization to only create the singleton when it is needed.*
- *May be a fancy global variable, with all that that entails.*
- *It may be difficult to customize creation of the singleton.*
- *What happens if/when the singleton "breaks?"*