EECS 293
Software Craftsmanship
2019 Fall Semester

Programming Assignment 10

Due at the beginning of your discussion session on

April 2-5, 2019

Reading

- Chapter 8 in Code Complete
- Singleton pattern (page 151 in Section 6.3, Code Complete),
- Try-finally (page 404 in Section 17.3, Code Complete)
- Items 69, 70, 73, 74, 75, and 76 in Effective Java
- Section 19.6 in Code Complete.

Grading Guidelines

An automatic C (or less) is triggered by an incomplete error-handling document.

Points will be deducted if code and branch coverage is incomplete. An automatic C (or less) is triggered by:

- Any routine with complexity greater than 4,
- Any substantially repeated piece of code, or by
- Improperly named routines.

Programming

In this assignment, you will develop the software that scans channels in a secure peer-to-peer network. The protocol's objective is for two wireless interfaces to agree on a channel to use for communication. In short, each radio scans through the channels, pausing briefly on every channel on which communication is detected. If it hears a call from another radio to its own address, it will stop on that channel and establish a connection. Although the protocol is encrypted, in this assignment, you can assume that the data stream has already been decrypted and presented to your protocol in plaintext. Your job is to develop the component that listens to a channel and decide whether to connect or not.

Every wireless interface has an address, which is a byte array. The address length is unspecified and can vary across interfaces. To establish a call, the caller will transmit a sequence that starts with the address of the intended receiver followed by the caller's. The simplest type of call is one between two interfaces whose address is a single digit, and looks like:

```
TO 1
TO 1
THISIS 4
```

The sequence represents a call from interface 4 to interface 1. The recipient address is repeated exactly twice. If either endpoint has a longer address, a more complex sequence is used:

```
TO 1
REP 2
REP 3
TO 1
REP 2
REP 3
THISIS 4
REP 5
```

The sequence represents a call from interface 45 to interface 123.

There is an additional problem, though. Assume that 45 wants to establish a connection to 123. Interface 123 is repeatedly scanning through many channels, spending a few milliseconds on each channel to see if there is an incoming call on that channel. If 45 wants to use a channel for the call, it must transmit long enough on that channel so that radio 123 has time to go through all channels and still

be in time to hear the call from 45. To solve this problem, radio 45 has the option to repeat the first sequence element (TO) thus giving radio 123 the opportunity to hear the first digit of its address. Other interfaces like 1 and 167 will also stop to listen but they will continue scanning once the full address follows. Then, the calling sequence becomes:

```
TO 1
TO 1
TO 1
TO 1
TO 1
TO 1
TO 1
TO 1
TO 1
TO 1
REP 2
REP 3
TO 1
REP 2
REP 3
THISIS 4
REP 5
```

The initial `TO` can be repeated between one and ten times, and the exact number of repetitions is at the discretion of the caller. The remaining `TO` and `REP` are not repeated.

Your job is to develop a component whose input is a receiver address, a local address, and an `InputStream` and that returns both a boolean variable that denotes whether to connect on this channel and the caller address. The input stream is supposed to contain repeated transmissions of the calling sequence as detected on a channel. However, since the receiver may tune in halfway through the transmission, the stream may start part way through the calling sequence. Furthermore, due to electromagnetic interference, some lines may be garbled. Your component should return at the end of the first useful calling sequence or at the end of the input, whichever comes first. Your code should process the stream incrementally without waiting for the end of the input.

The assignment does not prescribe any particular method for dealing with incorrectly formatted inputs, since it is your job to design and implement an error handling approach. However, your project must

implement a degree of robustness. Conversely, plain correctness will not be considered adequate. For example, your code should be robust to garbled lines, missing newlines, and many other potential errors. An especially robust approach will receive extra-credit.

Design Document

Submit a separate text file to document the error handling architecture that you will follow in your implementation. The architecture document should describe your error handling choices such as a strategy for implementing robustness and handling erroneous arguments, decisions on local or global error handling, error propagation through the code, presence and location of a barricade, and the other factors in the defensive programming checklist at the end of chapter 8. You will be asked to demonstrate that your code follows the error handling architecture.

Run

Create a new task called run (invoked with 'ant run' or 'make run') that executes your code taking the input from standard input and putting its output on standard output. Example of inputs and corresponding output values are posted on canvas. The input should contain a line for the local address, a line for the receiver address, and the input stream. The output should contain either the string "false" followed by an optional error message, or the string "true".

Since the run task will be used to test your code automatically, it is essential that it adhere closely to the input and output format.

Group Assignment

Your discussion leader will randomly group the section into two student teams. The team will be jointly responsible for Programming Assignment 10. In addition to the regular submission, you are required to post a private comment on canvas on the group dynamics, and in particular on the percentage effort that you feel was put together by the individual team members.

General Considerations

Your code should have an exhaustive unit test suite. Your code should have a reasonable number of comments, but documentation

is going to be the topic of a future assignment. As a general guideline at this stage of the course, comments should be similar to those accepted in EECS 132.

Discussion Guidelines

The project discussion will focus on defensive programming. In particular, you will be asked to demonstrate that your code follows the error handling architecture.

Submission

Create a repository called channels.git where you will post your submission. Make small regular commits and push your revised code and test cases on the git repository. Submit a separate text file to document the error handling architecture.