Data Structures and Algorithms

Report: Wireless Local Area Network Simulation

The aim of this coursework was to create a simulation of how laptops can connect to a WLAN. Since it is a simulation, the implementation is far more simpler than the actual process. In this practical, a laptop user can connect to the network after the server assigns an IP address to the laptop. Apart from this, the program should also meet more requirements like ping, disconnect, add and remove users etc. as well as any additionally needed specifications.

Classes

In order to design the solution for the given problem, it is initially required to decide on what should be created and how the solution should be implemented. Concerning the structure, we created three classes, one for the User, one for the Laptop, and a Program which represents the server and also contains the main. A User is identified by username and password and also has a laptop which has a hostname that is set by the user and an IP address set by the server when connected to the network.

While we first decided to have a Server class apart from a Program class with a main, we noticed that we had no attributes to store for it. It only contained the three data structures we had, and methods to obtain those data structures. While it made a few things easier by making the Program (main) class less cluttered, it ultimately lead to us attaching the Server class as an argument for every method and then using the methods from it in the same way we would use them if they were just static methods below the main. Additionally, there was no reason to ever have more than one server active at a time as the rest of the program didn't support multiple servers anyway. As such, with only one instantiation and a lot of redundancy added, we felt it would be more appropriate to just have its methods as part of the Program.

Data Structures

The program needs to keep track of data such as the number of added users as well as the number of laptops that are connected to the network. For this purpose, we chose to create two dynamic referenced based lists: a user list and a list of currently connected laptops. We decided on this structure due to the fact that the program needs to have access to all of the objects that are stored in both of the lists. However, the server also has a "bank" of available IP addresses that are stored in a queue. When a laptop wishes to connect to the network, the server dequeues (if available) an IP address and assigns it to the laptop, and if disconnected, the server enqueues the laptop's IP address back to the bank of IPs.

"Ping"

The interface of the "Network User" menu was upgraded by creating an additional option to the "Ping Computer" action. Since the user is allowed to choose whether to ping by hostname or IP, the user needs to specify which option is chosen. For the purpose of this, two different methods were created that correspond to each of the previously mentioned options and display (if applicable) the "pinged" laptop together with its IP.

"Start Server"

We assumed that the server needs to be started before choosing any other option, since the content from the input file should be read. For this purpose a static variable "launched" was created that indicates if the server is already started. In the case when the user chooses an option without initially starting the server, an appropriate message is displayed.

ADT additional methods

The created program needed extra methods for the reference based list class. The added methods that were implemented are the print method, add sorted, compares to items.

The print method prints every single element of the list. This was needed for the case when the user chooses the option “View Computers”, but also works for every other referenced based list.

The add sorted adds the object in a sorted manner, without specifying the index. If the object is a string it is added in an alphabetical manner. The method takes as an attribute a comparable object and if the given object’s letters are positioned higher in the alphabet the method returns a positive number. Otherwise, if the given object’s letters are positioned lower in the alphabet it returns a negative number.

The comparesToItems() is used when it is needed to check if two objects have the same value.