Design and implementation

Design:

When designing this solution, I identified two key data objects that would need to be stored together during this program’s operation. These were for the results from attempting to form a socket connection with various ports, called “ScanResult”, which contains all the data extracted from the Port / Socket scanning functionality of the solution. This data object is stored in an ArrayList Object with the aim of using this structure to populate the table with the results of the Port/Socket scan, as well as saving this data to a file by extracting all the elements stored in the “ScanResultList”. Similarly, to the “ScanResult” objects, another identified data Object is the result of scanning for devices connected to the network “DeviceDetails”, the ArrayList called “DeviceList” fulfils the same functionality as the other ArrayList implemented in this solution, table population and saving data to file. The ArrayList data structure was especially appropriate for this task as there is no way to know the size of these data structures once the respective scans were completed, because of this, an ArrayList can be initialised without a size, allowing as many computed scan results to be stored as needed.

The design of the solutions layout / GUI was a more challenging task, for this I used SWT builder. This benefitted the design by using the native controls for all the available UI components available on the host device. Furthermore, SWT is well integrated with Eclipse, allowing a simple download of the libraries to begin development. During the Design phase of development, I identified several widgets/components that would be needed for this type of solution, buttons are used for the majority of User input controls and a table widget for displaying the data stored in the ArrayLists. Additionally, a textbox would be needed for some input, possibly entering the file path for the source and destination folders used in the saving/encrypting/decrypting functionality of the solution.

In this solution I have implemented both static and non-static classes and methods. This combination stems from the utilisation of provided tutorial code such as the ScanResult class. My personal preference would have been to try to implement this functionality in a non-static way, trying to attempt a more object-oriented approach. However, as there is no mention of best-practice/object-oriented techniques in the task brief I chose to implement this combination as best as I could. The result is slightly difficult to read but functional code, which will be explored further in the Implementation section.

The brief requires the solution to identify port and socket information. Port information comprises of the local and remote Port numbers for the socket connection. From this we can infer more information, as for this program to return any socket/port information it means that the outputted port number is open and listening. While there are conventional services provided on typical port numbers, I determined it was not feasible to extract this information in this solution. The new socket created as a result of the connection of this code to the listening port is an application layer construct from which this solution extracts:

* Remote and Local Address
* Remote and Local Domain Name
* Remote and Local Port numbers
* Port status(Open/closed)

The brief further requires devices connected to the network to be listed. As such a device data object will be implemented to store the information this solution can extract from the network. Because of this, the inbuilt methods of the InetAddress class will be implemented in order to ensure the solution extracts the maximum amount of data possible.

Due to the solution being exported as an executable jar file, the IP address of the network to be scanned will, by default, determine the current networks IP address and scan for devices and ports on it. As such this does not need to be accounted for during this design phase.

Implementation:

This section will detail the actual implementation methods and classes used in this solution, any differences from the design will be explained and justified or critiqued where appropriate.

Firstly, the use of ArrayLists as opposed to a regular java Array has already been explained. There was an attempt to store both “ScanResult” and “DeviceDetails” objects as string representations together in one string ArrayList. However, due to the data needing to be both written to a file and used to populate the table, this was not appropriate. Instead we used these two object ArrayLists to create temporary ones in methods such as “load table” and “save data”. This is to ensure that data integrity is maintained during program operation and to allow for any expansion on functionality to occur on these temporary ArrayLists and not the programs “master” ArrayLists.

The methods for the “master” ArrayList classes are used to return the ArrayList itself, iterate over every item in the ArrayList and return it individually and finally to return the String representation of each object. However, in implementing the code to add items from an ArrayList to the table, various errors were output when using the “master” ArrayList class methods. Because of this, the implementation uses the temporary ArrayLists to iterate over the items and add them to the table widget.

The code responsible for loading the table full of data uses a Boolean variable for each ArrayList, to ensure the data stored in the ArrayLists does not get loaded into the table repeatedly. It does this by being initialised to false then set to true after data has been added to the table.

While implementing this solution in SWT Builder there were several decisions to be made such as the layout of the GUI as well as passing the ArrayList objects through the hierarchy of main-open()-createContents(). This proved more difficult than accounted for in the Design phase, as many of the feature to include included many “exceptions” that had to be accounted for either in the form of “throws declarations” and “try-catch” methods. Keeping the objects for List creation in the main method and passing them as parameters through the hierarchy was the most suitable solution to ensuring they can be reached throughout the code. This was achieved through a trial and error approach and given more time this is an area I would like to improve upon.

Further Implementation choices regarding the SWT builder are the use FormLayout. While tutorial guidance indicated that the grid layout would be the most appropriate layout, I found this not to be the case as adding further widgets(buttons etc) to the design was difficult and moving code around in the source .java resulted in further errors and difficulties. Which is why the actual implementation utilised Form Layout. It is worth mentioning that for more advanced/professional solutions Grid Layout would be preferable to enforce a more rigid structure to the solution however, when prioritising tasks in the implementation, the layout was given a lower priority compared to functional tasks.

The algorithm I implemented to populate the table simply pulls String representations of the objects out of the ArrayLists and adds them directly to a row on the table, once again the implementation of correctly formatted table columns proved more time consuming than anticipated and could have benefitted from better planning in the design of this solution. However, for this level of application and without further direction from the brief, I believe it is more than adequate, as the data extracted from the network by the solution can be easily and clearly read by the user before being saved to a file and encrypted.

This solution also implements the executor service interface represents an asynchronous execution mechanism which is capable of executing tasks concurrently in the background. This is used to speed up the process of scanning for ports by using multiple threads to compute the calculation of connecting Sockets in this application. As this particular task can be very time consuming this implementation is an excellent alternative to manually starting and closing threads in this solution.

The solution collects data using an implementation of the InetAddress class. When the program calls the method to attempt to create a “ScanResult” object a Socket is created using the current networks IP and the current Port number (based of the for loop in the portScanner() method). A timeout value is also passed in to ensure to connection doesn’t block(at least until a connection is established or an error occurs). This functionality returns useful information on the open / unsecure ports on the network, with more time the ability to scan ports along a specified range would be implemented, to allow the user greater control and flexibility when scanning for ports, as well as saving time on the scan.

Saving to file functionality is implemented in the form of a PrintWriter object, as well as a FileWriter. The alternative to this approach that was considered during designing was to use Serialisation of the objects then writing them to file. However, during the implementation it was discovered that these objects are written to file in Serialisation form and so not readable English. Once this was realised the PrintWriter method was implemented. The task brief wasn’t specific about the data in the file being readable, only that it must be saved and then encrypted/decrypted, yet the assumption was made that since the file was to be encrypted/decrypted anyway, the text must need to be readable, with this functionality used to protect the contents of the file.

The Encryption and Decryption code is implemented as an object which is called to encrypt and decrypt the file at the specified location. The main challenge here was to use the same generated “secret key” in both the encryption and decryption calls. The object is created in the even listener for the encryption button and as such will encrypt the file at the file location stored in the filePath variable. To implement the decryption code I attempted to pass a Boolean to the object to signal that encryption using the previously generated key is needed. However, this still left the issue of the “encryption/decryption” object being unreachable to for the code in the “decrypt” button. A hypothesised potential workaround to this issue could be to create the object with the necessary key in the main method, then pass this object through the hierarchy in order for it to be accessed by both the encrypt and decrypt buttons. But this was not tested due to time constraints arising from the difficulties in implementing the SWT Builder GUI.

For Encryption and Decryption, the encryption algorithm that was selected was the AES (Advanced Encryption Standard) which is a symmetric encryption algorithm that is more secure than the alternative DES (Data Encryption Standard), which is widely considered to be highly insecure and has been broken through Brute Force within a single day by machines in its history. This method was selected as the most appropriate as it is easily implemented and capable of handling upto 128 bit blocks using alternatively sized keys, the key size we selected was 128 bits, but this could be taken further by increasing this integer to 192 or 256 bits.