CST8130: Data Structures --- Assign #1- Router

Dynamically allocated array/Error and Exception Handling/Inheritance/Polymorphism/Files

DUE: Monday February 4, 2019 -- by 10PM SHARP!

You MUST use the default Package in your assignment so that I can compile and run your program easily. ALSO – all data members MUST be private in ALL classes (If you do not use a specifier, then the access defaults to public).

Reminder: As mentioned in class, we will NOT use the get/set design pattern in this assignment – note the methods listed to be written. You are allowed to use get methods (just not set methods).

Finally: Your program must be written using Object Oriented Principles (as outlined in the course notes)

Problem Description:

In this assignment, you will complete an object-oriented version of the software running on a networking router. Note you may NOT use the ArrayList class in Java for this assignment. You must use the classes as named in this assignment, but you can use different methods and data member names as long as your solution follows all Object Oriented Principles.

Background:

1. As we learned in CST8108 - Networking, A router is computer that runs software that is used to move packets (pieces of data) through a network. Not unlike letters in the mail, every packet consists of a destination address, a source address and contents (data). A router has a fixed number of ports to receive/send packets.
2. Each router maintains a table of destination addresses that it knows about, and which port (direction) to travel to get to that destination. So when a message or datat packet leaves its source address, it travels to the first router on its path. This router will look up the destination address in its routing table. If the destination address is found, then the packet is sent out the port entry for that destination in the table, which will result in the packet being sent and received at the next router. This process continues until the packet arrives at the destination. If at any point along the way, a router does not have a table entry for the destination address, then the packet is dropped immediately by the router.
3. Special packets are sent from each router to all of its neighboring routers, telling the neighbors of destination address that it knows about. (In effect, these packets say,"I know about such-and-such a destination, so add an entry to your table for that destination and send any packets for that destination to me"). These are called protocol or routing packets.
4. This description is simplified, and any of you with more detailed networking knowledge will recognize the simplicity. However, it is truly the basis for the processing on a router – and we will be able to produce a version of this software.
5. So, we want our program to read packets from a file – each line on the file will be a packet. This will simulate a data stream arriving on a router port or interface. Each packet will consist of
   - The type of packet - 'p' for protocol (this packet should be used to update the routing table) or 'd' (this packet should be routed and contains only data)
   - A destination IP address
   - A source IP address
   - Data – which consists of: the data associated with that type - string information about the port (ex e0) if packet is of type 'p' OR a string of data which should be routed if packet is of type 'd'
6. IP Addresses consist of four integer values between 0 and 255 – representing the four octets . I have written a class that handles this data type for you. You should not need to changes
7. Our program should build a table of destination addresses/ port to travel to get there. The required size of this table is unknown – but we will implement it as a dynamically allocated array of RoutingTableEntries of some arbitrary size and this is the important part of the assignment.

IPAddress class:

1. I have written this class – and it is available on BrighSpace as IPAddress.java. ***You can simply use the code I've given you – the most important thing you need to know is that you can compare two objects of type IPAddress to know if they are equal by using the isEqual method. You can use any of the methods described below to perform an action on objects of type IPAddress.***

2. This class models an IPv4 address has the following data members:
   - int address[4] – to hold each of the four octets in an IP Address, and
   - int subnet – to hold the subnet mask number of bits (CIDR notation).

In general, we assume the Subnet will be either 8 (for class A), 16 (for class B) or 24 (for class C).

3. The methods that are included with a brief description of what each should do: (Don't hesitate to ask me if you have any questions about any of the methods). Note – it is possible that not all methods will be used in your solution..some will be used only in your testing.
   - Default constructor
   - *initialize* - with int array [4] and subnet parameters
   - *initialize* - with object of type IPAddress
   - *readFile* - reads IP address and subnet info from parameter Scanner object parameter - returns boolean whether valid IP address was read (note the file object will already be open when this method is called, and is not be closed in this method)
   - *readKeyboard* - reads valid IP address info from keyboard
   - *isValid* - returns true/false if IP address is valid (each octet in valid ranges, and subnet info matches first octet)
   - *toString* – outputs an IP address in CIDR notation to a String
   - *getNetwork* - returns appropriate network address in IPAddress parameter (because there is too much information to pass back just through the return value) – note this will be used for method isEqual.
   - *isEqual* returns a boolean and bases the decision on if the two objects are on same network (have same network number – which of course depends on the subnet mask)

RoutingTableEntry Class:

- This class models a single entry in the routing table. It will have the following data members:
  - An *IPAddress* object to hold the destination address
  - *String* object to hold the portCode (for example e0, e1, s0/0/0 or s1/0) – see router class next below

- The actions that will be needed in this class include:
  - Default constructor
  - *addEntry* – to update the information in the current object from the parameters passed into the method
  - *toString* – to display this entry to a String
  - *searchForPort* – to compare the destination IP Address in parameter to this entry to see if they match, and if they do, send back portCode; or send back "" (empty String) if they don't match. This will be used to process data packets in the input and determine which port they should continue out of.

Packet Class:

1. This class will model a generic (base) packet. It will have the following data members:
   - Object of type *IPAddress* for the destination address
   - Object of type *IPAddress* for the source address
   - *String* object for packetData

2. Methods in this class will include
   - Default constructor
   - *readPacket* from Scanner parameter passed in, returning a boolean to indicate good data was read (or not)
   - getDestNetwork – which returns the network IPAddress of the destination address
   - *getPacketData* – which returns the packetData String
   - *processFoundPacket* – which does the action needed if destination IP address of the packet is found in the routing table
   - *processNotFoundPacket* – which does the action needed if destination IP address of the packet is not found in the routing table

DataPacket Class:

1. This class will be extended from the Packet class. There are no additional data members needed in this class

2. Methods in this class
   - *processFoundPacket* – which does the action needed if destination IP address of the packet is found in the routing table which is display a message which port the packet is exiting
   - *processNotFoundPacket* – which does the action needed if destination IP address of the packet is not found in the routing table which is to display a message that drops the packet

RoutingPacket Class:

1. This class will be extended from the Packet class. There are no additional data members needed in this class

3. Methods in this class
   - *processFoundPacket* – which does the action needed if destination IP address of the packet is found in the routing table which is display a message that the destination network is already in the routing table
   - *processNotFoundPacket* – which does the action needed if destination IP address of the packet is not found in the routing table which is to display a message that the destination network and data are being added to the routing table. Note a return value from this method alerts the calling method to do this *addEntry* action

Router Class:

This class will be the final class needed. It is going to represent the router. It will have the following data members:

- A routing table consisting of a dynamically allocated array [ ] *RoutingTableEntry* objects
- An *int* number of entries currently in the routing table.
- An *int* maximum number of entries currently allocated to the routing table

2. Methods in this class will include:
    - Default constructor - this will simulate the router being "booted" and will prompt the user for how big a table to allow, and update max number of entries and allocate this number of entries to the routing table.
    - *displayTable* will display the current routing table
    - *processPackets*– this is the method that will do the bulk of the router processing. It will have a parameter of a packet It will first, try to find the destination address in the existing table. It will then call the appropriate method – processFoundPacket or processNotFoundPacket in the appropriate Packet class (using polymorphism – which means you do NOT need to know if this packet was 'p' packet or a 'd' packet at this point).

Sample file:

p 192 168 1 2 24 192 168 1 4 24 e0
d 192 168 1 2 24 192 168 1 4 24 123456778123
p 192 168 4 1 24 192 168 3 2 24 e1
d 192 168 5 1 24 192 168 3 2 24 aaaaaa
p 192 168 4 1 24 192 168 3 2 24 e1

Processing for this file (line by line): (blue are my comments to help you understand)

Add 192 168 1 0 /24 e0 to routing table (destination not found – so it is added to table with the port in the data part of packet)
Data being routed out e0 is : 123456778123 (destination found – so routing message to port in the table for that destination)
Add 192 168 4 0 /24 e1 to routing table (destination not found – so it is added to table with the port in the data part of packet)
Packet to 192 168 5 1 24 is being dropped (destination not found – so dropping packet message)
Entry is already in table (destination found – displaying entry is already in table)

FINALLY, write ***function main*** which will:

- Declare a router object
- open a file (from user entered filename), process each packet (one line in file for each packet) and then close the file. Packets in the file have the following format:
    - **packetType** – char – 'p' for protocol (this packet should be used to update the routing table) or 'd' (this packet should be routed and contains only data)
    - **destination IP address and subnet number** (see *ReadFile* method in IPAddress class)
    - **source IP address and subnet number**
    - **packetData** - if packet is of type p – two char code for port (ex e0) OR data (if packet is of type 'd') – a string of data which should be routed out the port found in the routing table
    - **It would be useful to display the final routing table before your program ends!**

Submission:

You must submit to the assignment link in BrightSpace by the due date and time a zip file (named LastnameFirstNameAssign1) containing:

- all source code – ie .java files (Note – I may choose to re-compile your program….so all code must be available to me) with headers (see my header in IPAddress class)
- all class files

Failure to provide any of the above will have an effect on your grade for this assignment. Marking guide will be published shortly.

Hints:

This assignment needs to be tackled in a structured fashion in order for it to be finished quickly. Do not write more than 20-30 lines of code at a time without running your program. Start with opening and reading from the file to make sure that is working. Then work up in layers from there. Enjoy**!!**