COMPSCI 230 Assignment 2 (2019 S1)
Network Packet Transmission Visualizer

Instructor: Paramvir Singh
Course Coordinator: Angela Chang

Learning outcomes of this assignment

In completing this assignment, you will gain experience in:

- Designing the class structure for a Java GUI application with Swing components (a menu, radio buttons, a combo box and a JPanel) and AWT graphics (a diagram visualising data)
- Using a JFileChooser
- Handling various events associated with Swing components
- Data input from a text file

Depending on your design, you may also be using:

- Regular expressions
- Polymorphism with an abstract class and other OO features
- Exposure to Set collections (not taught in lectures)

This maps to the following learning outcomes of COMPSCI 230:

- *OO Programming:* describe and use the features typically offered by an object-oriented programming language, including support for classes, visibility, inheritance, interfaces, polymorphism and dynamic binding
- *OO Design:* explain and apply key design principles of object-oriented software development, including separation of concerns, abstraction, information hiding, programming to interfaces, coupling and cohesion, resilience to change, and reuse
- create simple OO design models
- *Frameworks:* describe important concepts of programming frameworks, including APIs, inversion of control and event-driven programming
- use a framework to develop a multithreaded GUI application
- *Concurrent Programming:* explain and apply the principles of application-level multithreading: threading, condition synchronization, mutual exclusion, and primitives associated with these

This assignment will be worth **5% of your final mark**.

Note: You can attempt up to 120 marks in this assignment, but you will only require any 100 of these to obtain full marks (i.e., there are 20 bonus marks that allow you to make up for marks lost in other parts of the assignment).

Due date and submission

Due: **11:59 pm on June 02, 2019**

Submission: **via the assignment dropbox** https://adb.auckland.ac.nz

What to submit: **all `.java` files that make up your application.** Do not submit as a ZIP file.

Introduction

In this assignment, you will work on a real life research problem. Our network lab hosts the Auckland Satellite Simulator (http://sde.blogs.auckland.ac.nz/). We use it to simulate Internet traffic to small Pacific islands that are connected to the rest of the world via a satellite link.

To do this, the simulator has a number of machines ("source hosts") on the "world side" of the simulated satellite link that transmit data in the form of small chunks of up to 1500 bytes called *packets*. These packets travel via a simulated satellite link to the "island side". Once on the island side, each packet ends up at a machine there. These machines are called "destination hosts".

The simulated satellite link delays packets and occasionally throws some away when there are more packets arriving that it can deal with at the moment. When the link throws packets away, the source hosts respond by sending less data for a while before attempting to send more again.

On the island side of the satellite link, our simulator eavesdrops on the incoming packets. It stores summary information about each packet in a *trace file*. The trace file is plain text and each line contains the record for exactly one packet (more on the file format in the next section).

What we would like to be able to do is get a graphical display of how much data comes from a particular source host over time, or how much data goes to a particular destination host over the course of an experiment. Experiments typically take between 90 seconds and about 11 minutes.

This is where your assignment comes in: You are to build an application that displays this data.

In the next section, we'll look at the structure of our trace files, before we take you through the steps for building the application.

Trace files

The total size of a trace file can vary substantially depending on the number of hosts involved in an experiment and the length of the experiment. On Canvas, there are two trace files for you to experiment with: a small one with 148315 lines, and a large one with 651274 lines. You can open them in text editors such as Notepad++. And yes you can edit them, too!

To get a good idea of your "typical" line, scroll down a good bit – the lines/packets at the start and at the end are a bit unusual. The following shows a bunch of typical lines from the large trace file:

```
108860  128.879102000  192.168.0.24   47928  10.0.0.5   5201  1514  1500  1448  …
108861  128.879885000  192.168.0.24   47928  10.0.0.5   5201  1514  1500  1448  …
108862  128.880603000  192.168.0.24   47928  10.0.0.5   5201  1514  1500  1448  …
108863  128.881481000  192.168.0.15   8000   10.0.1.25  59590 66    52    0     …
108864  128.881481000  192.168.0.9    8000   10.0.1.15  42081 66    52    0     …
108865  128.881481000  192.168.0.24   47928  10.0.0.5   5201  1514  1500  1448  …
108866  128.881495000  192.168.0.15   8000   10.0.1.25  59590 66    52    0     …
108867  128.882148000  192.168.0.3    8000   10.0.1.4   55442 1514  1500  1448  …
108868  128.882905000  192.168.0.3    8000   10.0.1.4   55442 1514  1500  1448  …
108869  128.883800000  192.168.0.3    8000   10.0.1.4   55442 1514  1500  1448  …
```

The lines here have been truncated to be able to accommodate them on the page, but they show all the data you will need. Each line consists of a number of fields separated by a single tab character. Note that fields may be empty, in which case you get two successive tab characters.

The ***first field*** on the left is just a sequential number for each packet that is added by the eavesdropping program. The ***second field*** is a time stamp that is also added by the eavesdropping program. Each trace file starts with a time stamp of 0.000000000 for the first packet in the file. You will need to use the time stamp column in order to find out what maximum value to scale the x-axis in your application to.

The ***third field*** in each line is the IP address of the source host. IP addresses identify machines on the network and help routers forward packets between source and destination. Each IP address consists of four decimal numbers between 0 and 255 separated by dots (full stops).

The trace files here only show packets heading towards destination hosts on the island side, so all source host IP addresses start with "192.168.0.", indicating that they are "world side" addresses.

The ***fifth field*** is the IP address of the destination host from the island network. All of the island addresses start with "10.0.". You will need the third or the fifth field to populate the combo box depending on the status of the radio buttons.

The ***fourth*** and the ***sixth field*** are the TCP ports on the hosts that the respective packets travel between. They identify the applications that have sent or would have received the packets, but are not relevant for your assignment.

Fields ***seven***, ***eight*** and ***nine*** are packet sizes in bytes. The size we're interested in here is that in **field eight**, it's the IP packet size. The size in **field seven** is that of the whole Ethernet frame that contains the IP packet, and **field nine** is the TCP payload size (the size of the content of the IP packet).

There are also a number of ***additional fields***: various flags, packet sequence and acknowledgment numbers, which are all irrelevant for your task. *You only need to look at four fields: time stamp, source and destination IP addresses, and IP packet size.* Note that some packets are not IP packets, meaning that the IP packet size field can be empty.

Step 1: Getting started

You are expected to do **your own class design** for this assignment, but there are some fairly obvious parts to the design: Firstly, it's a GUI application with one window, so you'll need a JFrame. In all our GUI applications in the lectures, we have extended that JFrame by a dedicated subclass for the application. So you'll probably want to do the same here.

The Swing components are all associated with the JFrame, so you'll need to import the associated packages, configure the components and add them to the JFrame either directly or via their respective parent. The ListOWords and Album applications from the lectures/supporting material are good examples for how to do this. Don't forget to configure the ButtonGroup to link your radio buttons!

One exception is the JPanel that shows the coordinate system with the graph. As you'll want to use this with the AWT graphics, it's best to extend JPanel, so a more specialised subclass can take care of the drawing.

Start by placing all components visibly within the JFrame – you may wish to set the JPanel background colours to something a bit different so you can see where in the JFrame they end up being positioned. Remember that a larger vertical position value puts the component further *down* in the frame. Suggested values that work for me (but aren't compulsory):

- The JFrame has size 1000 by 500.
- I've put the JPanel with the radio buttons at position 0,0 (upper left corner of the JFrame's content pane) and made it 200 wide and 100 tall.
- The JPanel subclass with the graph is at 0,100 and is 1000 wide by 325 tall.
- The coordinate system is 900 wide and 250 tall, and the ticks have length 5. Labels are positioned based on their lower left corner. The labels on the x-axis in my sample application have a horizontal position 10 less than the tick and a vertical position 20 larger than the axis. On the y-axis, the labels sit 40 to the left of the axis and 5 below the ticks.

You can earn a total of **41 marks** in this step:

- JFrame subclass correctly started via a class implementing Runnable & using invokeLater(): **3 marks**
- Menu bar present, showing a File menu with the two required items: **4 marks.**
- Menu item "Quit" quits the application and Menu item "Open trace file" opens a JFileChooser: **4 marks**
- The two radio buttons are visible: **4 marks**
- The radio buttons are mutually exclusive and one is selected by default: **4 marks**
- The JPanel (subclass) for the graph is visible and shows a default coordinate system: **8 marks**
- The coordinate system has between 8 and 24 ticks on the x-axis: **8 marks**
- The ticks are labelled in intervals of 1, 2, 5, 10, 20, 50, or 100: **4 marks**
- The axes are labelled with "Volume [bytes]" and "Time [s]" respectively: **2 marks**

In order to get the marks for positioning, no components or labels must touch each other or overlap with other components or parts the coordinate system (i.e., the JFrame and its contents must look neat and tidy).

Step 2: Design your classes

Beyond the JFrame subclass and the JPanel subclass, this is really up to you. You could use classes for the trace file, the hosts (even source or destination hosts), lines, packets, graph elements – you decide where you want to have your data and where you want to accommodate your functionality. You need to design and use at least two extra classes, but you will probably want more than that.

There are no marks in this step, but it is a precursor to Step 5, which you should complete at the end once your design is final and you have implemented as much of it as you could.

Step 3: Extract the source and destination hosts from the trace file

First ensure that the combo box is invisible before you open the first trace file.

Using the trace file selected by the JFileChooser, parse the file to extract the source and destination hosts. Check which radio button is active and show the respective set of hosts selected in the combo box.

Hint: There are a few little catches here:

1) Splitting lines can be done with the `split()` method of the `String` class, which takes a regular expression as its parameter.
2) The lists must not contain duplicates, so you can't simply add each line's host entry to an ArrayList and then load that ArrayList into the JComboBox. Instead, Java has a data structure quite similar to an ArrayList that doesn't allow duplicates: the HashSet class. It implements the Set interface, and for Strings, you can use it as follows:

```
import java.util.Set;
import java.util.HashSet;
…
Set<String> myUniqueStrings = new HashSet<String>();
…
myUniqueStrings.add(aStringWeMayOrMayNotHaveSeenBefore);
```

   If `aStringWeMayOrMayNotHaveSeenBefore` is not yet in the hash set `myUniqueStrings`, it will be added. If the string is already in the set, the add() method does nothing.

3) The lists must only contain IP addresses of IP version 4 (IPv4). Some packets in the trace file are **not** IP packets and the entries for source and destination in the respective lines will be empty. You must skip these packets. Think regular expressions, perhaps.
4) The lists in the combo box must be sorted. To do this, you need to figure out how to sort IP addresses in Java. This can be done in a number of ways. I use *Collections*.sort() and store the IP addresses in a class that has an appropriate compareTo() method. Google is your friend!

Once you have accomplished this, ensure that the combo box updates every time you click the other radio button and every time you open a new trace file. For this, you need to:

- Implement and add the necessary event handlers for the radio button.
- Implement the code required to (where applicable) fetch the list of hosts (should you generate these lists once when you load the file, or each time the radio buttons change?).
- Implement the code required to wipe and re-populate the combo box. Look to the Album example from the lectures/supporting material for a guide here.

Ensure that you get the correct list of hosts in the combo box each time.

You can earn a total of **36 marks** in this step:

- The combo box is invisible before the trace file is selected and opened: **2 marks**
- The application shows some evidence of a trace file being opened and its contents being processed after one selects a file with the JFileChooser (e.g., plot appearing, combo box getting populated with sensible data): **8 marks**
- The combo box becomes visible at this point in time: **3 marks**
- The combo box contains a list of IP addresses: **5 marks**
- The list contains the selected type of host addresses (192.168.0.x if source hosts are selected, 10.0.x.x for destination hosts): **4 marks**
- The list does not contain duplicates: **5 marks**
- The list is sorted correctly: **4 marks**
- The list updates correctly when the selected radio button is chosen: **2 marks**
- The list updates correctly when we open a new trace file: **3 marks**

Step 4: Compute and plot the data

Implement the necessary code to:

- Compute the total number of bytes that the selected source/destination host sent/received for each 2 second interval (or 1 second interval if you wish).
- Compute the maximum number of bytes across all of these intervals.
- Re-draw the coordinate system with appropriate ticks and labels for the whole duration of the experiment. "Appropriate" means that the horizontal axis must be scaled to cover the time period of the experiment within at least one tick, that the vertical axis must be scaled so its maximum must be at most one tick above the maximum number of bytes, and that the number and format of the ticks on each axis must conform to the requirements of Step 1. The vertical axis should have between about 4 and 10 ticks – again the appropriate maximum number here depends on the requirement that the labels should not touch or overlap.
- Plot the bytes data from the first bullet point in a different colour, either as a bar plot (as in the video), as a curve with lines, or as a series of markers.
- Ensure that the plot updates each time you select a new host, change between source and destination hosts, and open a new trace file.

You can earn a total of **38 marks** in this step:

- The application shows a data plot of sorts: **8 marks**
- The data plot is actually correct (shape-wise) and corresponds to the host selected: **8 marks**
- The x-axis scales correctly: **3 marks**
- The x-axis has an acceptable number of ticks (8 – 24): **4 marks**
- The x-axis is labelled correctly: **4 marks**
- The y-axis scales correctly: **3 marks**
- The y-axis has an acceptable number of ticks (4 – 10): **4 marks**
- The y-axis is labelled correctly: **4 marks**

Step 5: Document your classes

Document your design. The documentation must consist of:

1) "Javadoc" style comments for each public non-inherited method in your code (except for event handlers in anonymous classes). See "Writing Doc Comments" under:
http://www.oracle.com/technetwork/java/javase/tech/index-137868.html
Your comments only need a description of what the method does and the @param and @return block tags.

You can earn a total of **5 marks** in this step:

- *Javadoc comments:* **5 marks** (minus one mark for each un-/incorrectly documented method)

Debugging

I strongly recommend that you use Eclipse for this project. One particular challenge is the parsing of the trace file in Step 3 and (depending on your design) Step 4, as well as the scaling and plotting operations in Step 4. Much of the intermediate operations here happen behind the scenes – there is no visible output till the very end of a lot of these operations and many things that can potentially go wrong. One nice feature in Eclipse is that you can still use `System.out.println()` etc. to write to the console, so you can add such statements to check that your code behaves as expected. You'll also get to see any stack traces when exceptions are thrown. And of course, you can add breakpoints and debug using Eclipse's debugging features!

Notes

This really shouldn't be necessary, but **note the comments at the bottom of Assignment 1 on originality of work** that you submit & preventing others from using your code.

Not everything that you will need to complete this assignment has or will be taught in lectures of the course. You are expected to use the Oracle Java API documentation and tutorials as part of this assignment.

Post any questions to Piazza or take them along to the COMPSCI 230 labs/lectures – this way, the largest number of people get the benefit of insight!

I may comment along the lines on Piazza to deal with any widespread issues that may crop up.

As an additional challenge: As long as the functionality / specifications above are preserved, you are welcome to add any additional features that may be useful, e.g.,

- The ability to look at flows between a particular source and destination host
- The ability to look at flows from/to/between particular ports
- The ability to save or print the graph / data in the graph, etc.

Any such additional features do not attract marks but would certainly count in your favour should you find yourself in the unfortunate position of needing an exam aegrotat, or the fortunate position of needing a recommendation letter from me.