Networks
Project 1c
Flow control

This assignment is an extension of Projects 1a and 1b, but the code has undergone enough transformation that your solutions to that project won't be something that you want to copy-and-paste. Instead, you will be given an existing `ParityDataLinkLayer` from which to work, and to which you need to add flow control capabilities.

1 The simulator

Getting it: The code from which you are working is similar to the previous code from Projects-1a and -1b, but it's different enough that it will require some time to get acclimated.

To get started, download the simulator's source code:

`https://bit.ly/cosc-283-23f-p1c`

What changed: The general layout of classes is unchanged, and the `Medium` abstract class and its descendents are unchanged. However, nearly everything else is altered. Some of the changes you must understand in order to know how to test and debug your code, but you will not need to alter what you see. Mostly, you will need to fill in a number of abstract methods in your data link layer in order to make flow control happen.

The changes to `Simulator`, `Host`, and `PhysicalLayer` are all to drive each host as its own *thread of execution*, running independently of any other host and of the simulator itself. Data is now buffered in the data link layer when provided by both the host (for sending) and the physical layer (for receiving). The buffered data coming into the data link layer isn't processed until an *event loop* in the thread sees a non-empty buffer and calls a method to process it. That is, most of what you need to grasp is in `DataLinkLayer` and its subclass, `ParityDataLinkLayer`.

Here are the methods to which you must pay the most attention:

- `go()`: This is the event loop method. Once it is called, it loops forever (or until the `stop()` method is called as the simulator ends), looking for data it needs to send, received data it needs to process, or timeout events. It triggers all of the code you will be writing.
- `sendNextFrame()`: Called by the event loop when the `sendBuffer` is not empty. It grabs up to a frame worth of bytes, and then:
  1. Calls `createFrame()` to generate the frame needed to transmit that data.
  2. Calls `transmit()` on the framed data.
- `createFrame()`: The roll of this method is largely unchanged. It is where you will need to add code for frame numbers, etc.

- `finishFrameSend()`: After a frame is actually sent, it is passed to this method (defined in `ParityDataLinkLayer`) so that it can manage the flow control of it. This method can buffer the frame, and record that the layer is now waiting for an acknowlegement of the frame. It may also record the time so that it can later determine if a timeout needs to occur.
- `receive()`: As before, it accumulates bits received by its physical layer. As whole bytes arrive, each is queued in `receiveBuffer`.
- `processFrame()`: Called by the event loop when `receiveBuffer` is not empty. As before, it returns `null` when it cannot find a complete frame. If it does find a frame, it must remove the tags and metadata from it, check it for errors, and then return the extracted data if all correct.
- `finishFrameReceive()`: This method is called whenever a complete frame is received. It is responsible for delivering the data to the client `Host`, and should then perform the task of sending an acknowledgment to the receiver.
- `checkTimeout()`: This method must be defined to check whether too much time has passed since a frame was sent, thus triggering a re-send.

2 Implementing Stop-And-Wait

Your assignment: Within the structure described above, implement *positive acknowledgment with retransmission (PAR)* (a.k.a., *stop-and-wait*) flow control within the data link layer. I recommend copying the provided `ParityDataLinkLayer` to a new subclass, `PARDataLinkLayer`, and build on the provided code as a skeleton.

Extra challenge: Implement *sliding windows*. I recommend using a window size of 1 frame on the receiver, implementing a simpler *go-back n* form of this protocol.

3 How to submit your work

Go to Gradescope for our course, where you can submit your work. It will be auto-tested, and you will see whether it *compiles* and *runs* successfully. Again, if the run fails, it won't tell you why; you need to go back and do more testing yourself. You may submit early and often!

Notice that **you should only submit** `PARDataLinkLayer.java`. *Don't submit the other source code or class files.* If you implement *sliding windows*, let me know, but don't submit it!

This assignment is due on Sunday, Oct-22, 11:59 pm.