NETWORKS
PROJECT 2
Stop-and-wait flow control

This assignment is an extension of Project 1. Here, you will modify/extend your data layer (you may use your `ParityDataLinkLayer` or your `CRCDataLinkLayer`), adding to it an implementation of *stop-and-wait* flow control.

1 The simulator

Getting it: Although this project is an extension of the previous one, it does require adapting your code to a slightly different code base. Later we will examine the changes from the previous iteration, but first, get the new code:

```
$ mkdir -p networks/project-2
$ cd networks/project-2
$ wget -nv -i https://bit.ly/COSC-283-project-2-source
```

Copy one of your data link layers from Project 1, something like this:

```
$ cp ../project-1/ParityDataLinkLayer.java .
```

What changed: Most classes are unchanged. Specifically, the `Simulator`, `Host`, `PhysicalLayer`, and `Medium` classes (and subclasses) are the same as in the previous assignment. However, there are significant changes to the `DataLinkLayer` abstract parent class, and these changes affect any subclass.

The most significant changes are to the **abstract methods** that subclasses must implement, although other methods have been changed as well. Here is a list of the methods from `DataLinkLayer` that have changed, and how those methods interact with one another:

- `send()`: It now buffers any data to send, and then repeatedly calls a new method, `sendNextFrame()`, to handle the sending of a single frame. Implied in this change is that `send()`, via this restructuring, now properly divides outgoing data into smaller frames of no more than 8 data bytes each.
- `bufferForSending()`: Copy data provided in a `byte[]` into an outgoing queue. Later used by `sendNextFrame()`.
- `sendNextFrame()`: See above. This method pulls up to 8 more bytes from the send buffer, and sends those data bytes as a frame. Notably, this method performs a number of significant tasks:
  1. Call `createFrame()` on as many as 8 data bytes pulled from the send buffer.

2. Call `transmit()` on the framed data. See below on a modest change to `transmit()`; here, we need only to know that it will send the frame, one bit at a time, through the physical layer.
3. Call `finishFrameToSend()`, whose job will be to process any acknowledgment response, advance the frame number, etc. More on this below as well.

- `abstract finishFrameSend()`: Process any acknowledgment frame that the receiver may have sent in return. Note, critically, that given the structure of the simulator, *the receiver will have completely sent any response before this method is invoked*—the acknowledgment, if it arrived, should have been buffered by receiving code. The method should also do things like control the advancement to the next frame number to send (if the send was successful).

- `transmit()`: In the previous version, this method accepted only one byte as a parameter. Its parameter is now a `byte[]`, where the method will advance through the bytes in the sequence on its own.

- `receive()`: As before, it accumulates bits received by its physical layer. It then performs a number of tasks, when appropriate:

  1. When a byte-worth of bits has been received and buffered, the byte is reconstructed and added to a queue. Note that this queue, previously named `byteBuffer`, has been renamed to `receiveBuffer`, providing symmetry with and disambiguation from the `sendBuffer`.
  2. As in the previous version, it calls `processFrame()`, which determines whether `receiveBuffer` now contains a complete frame, and if so, extracts and returns it.
  3. If a complete frame is detected and extracted, the extracted frame contents are passed to the new method `finishFrameReceive()` (described below).

- `processFrame()` : This method operates much as it did before, but for convenience, I changed it to return a `Queue<Byte>` of the extracted data, rather than a `byte[]`. Doing so had the data returned by this method match what `finishFrameReceive()` takes as a parameter.

- `abstract finishFrameReceive()`: Complete the receiption of a frame by checking its contents (e.g., perform error detection). If the frame is properly formed, then an ACK frame should be sent, the expected frame number advanced, and the frame delivered to the client (the `Host`); if the frame fails the error detection test, a NAK frame should be sent, and the frame contents discarded (ignored).

2 Implementing Stop-And-Wait

Your assignment: Within the structure described above, implement *positive acknowledgment with retransmission* (a.k.a., *stop-and-wait*) flow control within your data link layer of choice.

How you might go about it: There are a number of places in which you will need to change and augment your code:

- **Frame format:** You must now add, to your frame structure, a predicitable place to communicate essential metadata. Each frame should have a frame number. Additionally, each frame should indicate whether it carries data or an acknowledgment; if the latter, is the acknowledgment positive (ACK) or negative (NAK)? This metadata must be in addition to the parity/checksum value used for error detection.
- **Frame number tracking:** What frame number is the sender sending? What frame number is the receiver expecting? Be sure that your data link layer maintains this state, and that it advances the values at the right time.
- **Finisher methods:** The two new abstract methods, `finishFrameReceive()` and `finishFrameSend()`, are there respectively to handle the sending of ACK/NAK frames and their reception. You may want, in your implementation, to write helper methods on which these call. For example, it is likely that you will want a method for constructing an acknowledgment frame, and then have `finishFrameReceive()` call it.
- **The receiving code:** Receiving a frame now must depend on frame type. Specifically, `processFrame()` now must handle the reception of ACK/NAK frames, as well as the data frames it already processes.

3 How to submit your work

Submit your `ParityDataLinkLayer.java` **or** `CRCDataLinkLayer.java` source code files (whichever one you chose to modify for this assignment), using one of the two usual tools:

- **Web-based:** Visit the submission system web page.
- **Command-line based:** Use the `cssubmit` command at the shell prompt on `remus/romulus/vega.cs`.

This assignment is due on Wednesday, Oct-24, 11:59 pm.