NETWORKS
PROJECT 1
Data link error detection

For this project, you will handle some bit-inversion errors introduced during transmission. Specifically, we will simulate communications across different media, some of which introduce different types of error. Your task will be to **write a data link layer** that detects whether an error has occurred.

1 Getting the simulator

This project will require you to extend the capabilities of a simple point-to-point network simulator that is written in Java. The simulator code comes as a directory of Java code; you will need to compile and run that code from a command-line shell, providing arguments to the simulator as you run it. If you have appropriate tools (e.g., JDK, eclipse) on your computer, you're welcome to do the work there; if you don't have such tools, you may login to the college's server, `vega.cs.amherst.edu` to use the tools available there.[1]

To get started, create and change into a directory for this project, and download the simulator's source code:

```
$ mkdir -p networks/project-1
$ cd networks/project-1
```

Download the project's source code:

```
$ wget -nv -i https://bit.ly/COSC-283-project-1-source
```

You will obtain a number of source code files that, together, simulate a partial *network stack* (using only the layers we've covered so far); this stack is created for each of two hosts, connected by some (simulated) medium. You will be modifying this program by adding new subclasses to the `DataLinkLayer` class.

2 The parts of the simulator

The simulator provides two *media* (and I may add others, later):

1. `PerfectMedium`: Connect two hosts with no errors ever introduced. The user specifies `Perfect` at the command line to use this medium.

2. `LowNoiseMedium`: Connect two hosts with infrequent, uniformly distributed bit inversions. The user specifies `LowNoise` at the command line to use this medium.

[1]This server has a JDK for Java 7 installed, as well as *emacs*, *eclipse*, *netbeans*, and *sublime-text*, all of which you may use on this code. If there are other tools you want to use on this server, let me know, and I can try to install them.

A *physical layer* object connects directly to a medium. There is only **one type of physical layer**. It accepts a sequence of bytes which it then sends, one bit at a time, across the medium. The receiving physical layer reconstructs the bytes, one at a time, delivering each complete byte to its data link layer.

Currently, there is one implemented data link layer:

- `DumbDataLinkLayer`: This particular data link layer uses *start/stop tags* and *byte packing*[2] to frame any data that its network layer asks it to send. It creates a single frame for any sequence of requested bytes, no matter the length, and most critically, it performs no error management. To use this data link layer, the user specifies `Dumb` at the command line.

There is also a `Host`, of which there is only one type, that is the *client* of any data link layer. It drives the data link layer to send or receive messages.

The whole thing is driven by the `Simulator`, which creates two (simulated) hosts and their respective data link and physical layers, which it then connects to the (simulated) medium. It then triggers one host to send a message to the other, printing the outcome.

3 Running the simulator

After obtaining the code, you should be able to compile and run it. The simulator reads a file to determine what data to send, so you should create a text file with a small message in it. (For example, create a file named *message.txt*, with that file containing some short message to send.)

Once you have a message file to be transmitted, you must specify, on the command line, which `Medium` subclass and which `DataLinkLayer` subclass to use. You do so by providing the leading portion of the name of the subclass on the command line. For example, if you want to use the `DumbDataLinkLayer` with the `PerfectMedium`, you invoke the simulator like so:

```
$ java Simulator Perfect Dumb message.txt
```

The simulator will read the message file, pass it to one host to be sent, and then query the receiving host for what it received. It will then print the received data. If you try using the `LowNoiseMedium` to introduce some errors, you will see some characters get corrupted (and, perhaps, see the whole frame broken by the corruption of the *start* or *stop tags*. error

4 Writing new data link layers

YOUR ASSIGNMENT: You must create **two new data link layers** that are subclasses of the abstract `DataLinkLayer` class:

1. `ParityDataLinkLayer`: Use a single, simple *parity bit* to detect one-bit errors on each frame.

[2]That is, byte-based escape codes for data that happens to match tag values.

2. `CRCDataLinkLayer`: Use the CRC checksum method to detect errors on each frame.[3]

For each of these implementations, when an error is detected, print an error message, show the (incorrect) data, and do *not* provide the data to the receiving host.

Both of your layers **must divide each message into smaller frames** (unlike `DumbDataLinkLayer`). Specifically, each frame should contain **no more than 8 bytes of data** each. Be sure to test your code with longer messages to be sure that the data is being divided correctly.

How to add a new data link layer to the simulator: To add a new data link layer, simply copy the source code of one of the existing data link layer subclasses (e.g., `DumbDataLinkLayer.java`) into a new file of your own (say, `ParityDataLinkLayer.java`). Edit the file and rename the class, and then change the methods so that it detects/corrects errors differently.

Note that **you do not need to change `Simulator.java` or any of the other existing classes for the simulator to recognize your new data link layer.** You are welcome to look inside `Simulator.java`, as well as the `create()` method in `DataLinkLayer.java` itself, which uses the command line input to form the names of subclasses, and then applies *reflection* to create objects of those classes. Thus, so long as your subclasses have the right kind of name (e.g., the names of data link layer subclasses end with `DataLinkLayer`), then the existing code will use them just as they do the provided classes.

5 How to submit your work

Submit your `ParityDataLinkLayer.java` and `CRCDataLinkLayer.java` source code files 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`.

This assignment is due on Wednesday, Sep-26, 11:59 pm.

[3] Consult the text to select a good generator polynomial to drive your CRC. I will be testing your code by introducing a number of kinds of errors with media of my own making.