Homework Assignment 3

You are to read a simplified version of internet packet data and reassemble the packets into sorted order so that the underlying message can be read.

Packet

An individual "packet" will consist of a single line of input data containing a `messageID`, a `packetID`, a `packetCount`, and a `payload`. The first three of these are to be `int` variables and the last will be a `String`. The first two of these have the obvious meaning. The `packetCount`, is the number of packets in the current message. The `payload` is the actual text of the message that is being carried in this particular packet.

The `Packet` class is essentially the same thing as what we have been calling a `Record` class all through the semester, and an `interface IPacket` can be found on the website.

PacketAssembler

The work of the assembly of packets is to be done in a `PacketAssembler` class for which an `interface IPacketAssembler` is on the website.

You are to implement an `ArrayList` of `ArrayLists` for handling the packets (as in: `ArrayList<Arraylist<Packet>>`). The inner list will be a list of the packets for a given message. The outer list will be the list of the packet lists for each of the messages.

The packets themselves will come in in no known order, so you will see packets from different messages coming in one after another, and the packets for a given message will not necessarily come in in the order of their packet ID.

You will have to determine if a packet is from a new message ID. If so, then you will need to create a new `ArrayList<Packet>` for that message. If not, then you will do an insertion sort of the packet into its appropriate list so that the inner list is in sorted order of packet ID. You will need to ensure that you keep only one copy of each packet, so if the packet ID matches one you already have stored, you can toss the new packet. (Or if you wish you can write the new packet on top of the old one. Neither strategy works all the time, so either is acceptable.)

When a message is complete (defined as: when the number of packets stored equals the number of packets in the message), you are to print out the message and delete the message from the outer `ArrayList`.

Take a moment to look at the commented out private methods in the interface. The message IDs are not in sequential order, so you can't access the entire message by means of the message ID. (This is different from the packet ID, which can be used in a completed message to access the packet.) So you will probably have to use the message ID value in a given packet as a means to look up the subscript in the outer `ArrayList` of a given message.

COMMENT: Yes, you could do this a little differently. When you get the first packet for a new message, you could conceivably create a new `ArrayList<Packet>` and add as many dummy records as were indicated by the `packetCount` value. Then, instead of doing the insertion sort, you could simply store in the appropriate location. The big difference here would be that since you would always have a "complete" list, the only way to tell that you had found all the packets would be to run through the list to verify that no null/dummy values were left.

Driver

Note that the `Driver` code calls up two methods for the packet assembler. Since the outermost unit of information coming in is the packet, but we don't know what message the packet belongs to, we read packets in the driver and the `readPacket` method will have to do list manipulation if the packet read is the first packet of a new message. And if it's the last packet of a message, it will return the message ID so we know that we should dump that message. The `dumpMessage` method will also have to delete the message from the outer list after it dumps the message to the output.