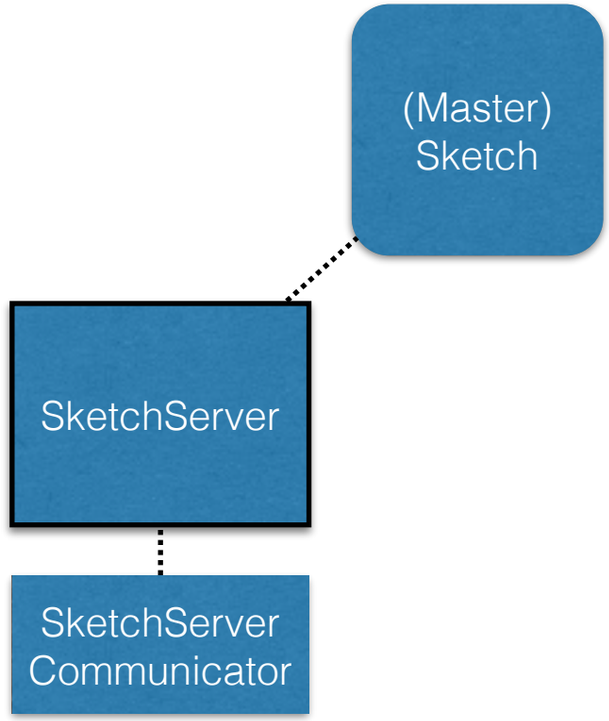
Server

At the heart of this whole architecture is a single instance of our SketchServer. Here, we've visualized the fact that a SketchServer is a component in it of itself, and that it manages (or, *has a*) its own (master) copy of (1) a Sketch, and (2) a ServerCommunicator.

A Sketch is basically just a collection of shapes (Rectangle, Ellipse, Segment etc.).

A ServerCommunicator is a subclass of the Thread class and enables communication between the SketchServer and an Editor.



Client

Here, we've shown the Editor. Similar to the SketchServer, an Editor is a component in it of itself, and it manages its own (local) copy of a Sketch, as well as its own EditorCommunicator.

An EditorCommunicator is also a subclass of the Thread class and enables communication between the Editor and the SketchServer.

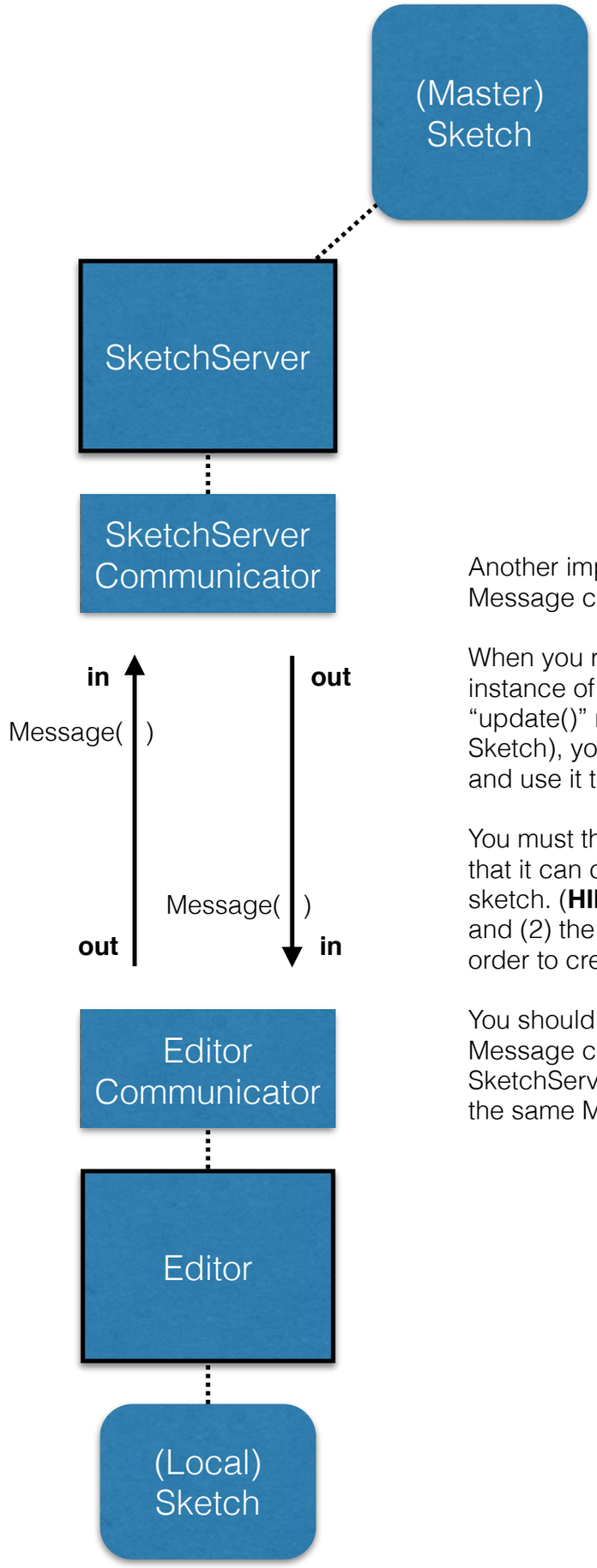
Note that it is important to understand that an Editor simply has a local copy of a Sketch, while the SketchServer holds the master copy of the Sketch. If we have multiple Editor objects, our only hope for keeping them in sync is to have one component managing the master copy, and telling others what their copy should look like.

If we ever want to perform some operation, we then "request" to have that operation done by sending a message to the server. The server's job is then to accept *valid* requests, make the changes (as requested) to the Sketch, and inform all connected clients (Editors) about the state of the Sketch.

Simple Example of Client/Server

This simple example shows an instance of a SketchServer that *has a* Sketch *and* a single ServerCommunicator instance. We also see a single instance of an Editor which *has a* (local copy of a) Sketch *and* a single EditorCommunicator instance. Communication between the SketchServer and the Editor happen between their respective "Communicator" objects.

I've also visualized the connection between the ServerCommunicator and EditorCommunicator via in/out labels. Recall that a Socket object is created "on both ends" so that a (Server/Editor)Communicator can create input/output streams. A ServerCommunicator object gets input from the EditorCommunicator's output and vice versa.



Another important component in this architecture is that of the Message class.

When you read a message in as a String, you should create a new instance of a Message and give it the String. By calling the "update()" method on a message (and passing it a reference to a Sketch), you should effectively parse the current message (String) and use it to update the Sketch as specified by the message.

You must think about how you will structure/format a message so that it can convey the necessary details to correctly update a sketch. (**HINT**: think about (1) the operations you must perform and (2) the information that each of those operations needs in order to create a message with all the necessary details.)

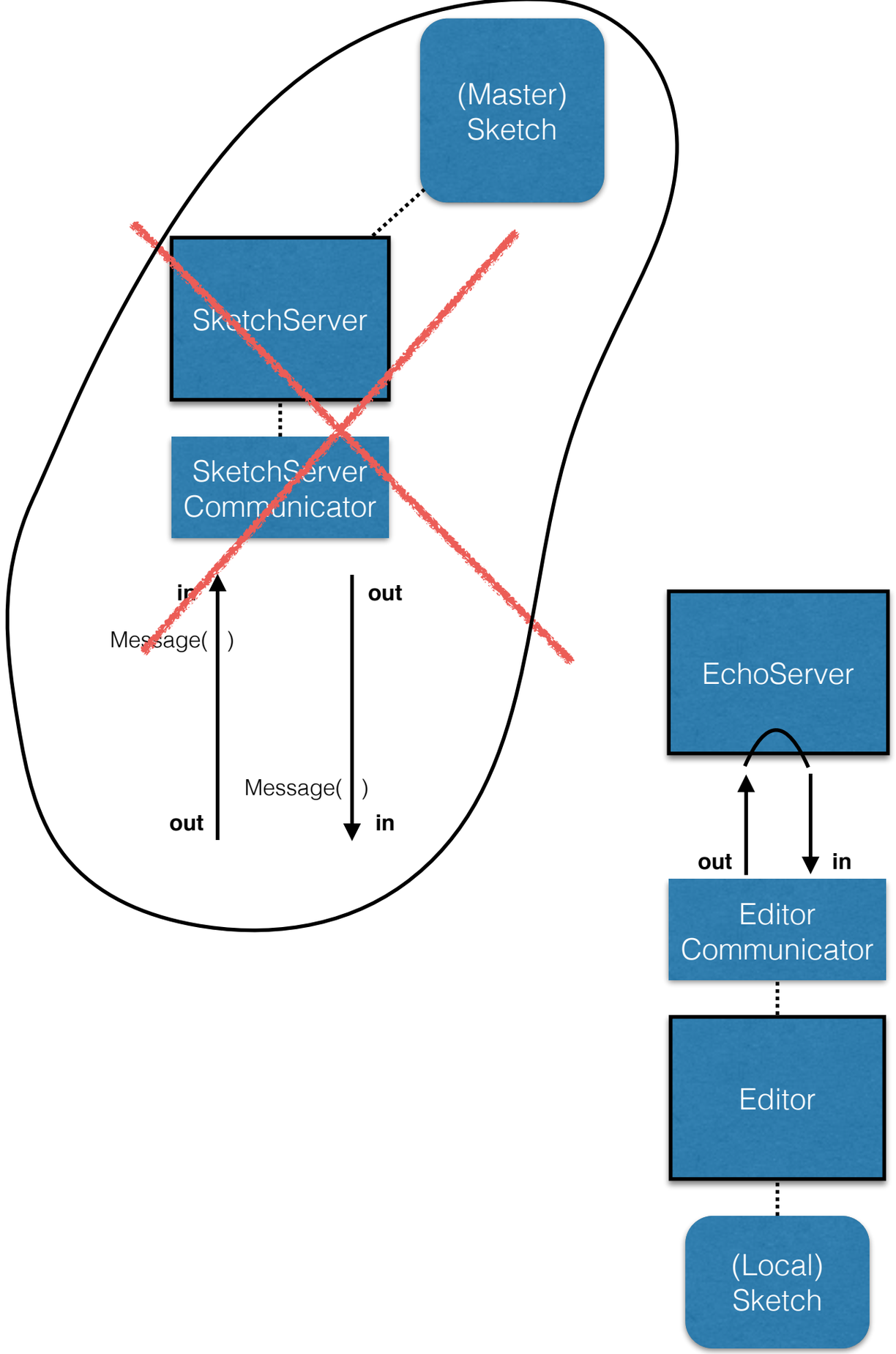
You should note that it is not necessary to have a separate Message class for the server *and* the client. Both your SketchServerCommunicator and EditorCommunicator should use the same Message class.

Development/Testing

To assist you in developing/testing the client-side part of this lab, I have provided an “EchoServer”. Rather than sending messages to a SketchServer and getting messages back from a SketchServer, you can use the EchoServer.

There are a number of ways that this is helpful:

- Your SketchServer is not fully implemented (specifically, your SketchServerCommunicator is not fully implemented). It is hard to know if this is working correctly if your Editor, etc., aren’t working correctly.
- An EchoServer allows you to act as if you are sending out messages to a server and getting messages back from a server. This is, in reality, all that is happening. However, the SketchServer may need to modify messages to correctly handle working with multiple clients. For initial development/test though, you are not worried about multiple Editors (clients), you just want to get one working! :)



Complete Collaborative Graphical Editor Architecture

In reality, the SketchServer can support more than one Editor talking with it. It does this by creating a new ServerCommunicator to communicate with each new Editor that connects to the SketchServer. Again, note that all communication between an Editor and the SketchServer go through their respective (Server/Editor)Communicators.

Here, we see that the SketchServer keeps track of all of its current ServerCommunicator objects by storing them in a list (ArrayList).