Abstract Design

ServerDocument (implements JSwing Document)
The ServerDocument is a class that will implement the JSwing Document (which works with TextArea). This design choice was because we noticed that the Document interface allows insertion of elements into a Position of the text, as opposed an index. This is appropriate for editing as insertion and deletion is done continuously. There is also a DocumentListener which would be appropriate for “listening” to changes to the Document.

There will only be one global ServerDocument. All clients will see and edit this global ServerDocument. There will be no local ServerDocument for each client (but there will be a local GUI), as this would mean that multiple versions of the ServerDocument will have to be merged, and edits may be overridden.
Using a global ServerDocument will be less problematic, erase the issue of overridden edits, and allow users to see what is being edited(especially with the way we have defined an edit to be).

Rejected Design Decisions

- Creating a Document Class, which contains the text as an ArrayList of characters. This was our original idea before we discovered Document. Rejected because the Document interface has methods that we need.

Server/Client
This is where the ServerDocument is stored.
We’ll be mapping the names of the documents to its appropriate Document instance.
Why do we want to store it in the server?
This means that as long as the server is running, any and all clients will be able to access the various documents and its edits. Even when there are no clients connected, as long as the server is still connected, the documents will exist.

Clients will connect to the server. The client will have a local GUI, a local cursor, but will edit the global ServerDocument. This is to prevent edits being overridden. Clients can disconnect and connect to the server and can still access the edited ServerDocument.

The only thing shared between clients is the ServerDocument. This is where we will have to be very careful with our thread safety. Everything else is local to the client and hence will be thread safe.

Edit
An edit class will be created.

Adding words:
An edit is an addition of a String in front of the client’s current cursor position.

As the edit is being made, an edit object will be added to the ServerDocument. This is so that we can lock this object -- only the client making the edit will be able to touch it. This would prevent simultaneous editing of the same thing.

An edit object will be unlocked and the edits will become part of the global ServerDocument when the following occurs: 1) spacebar is pressed 2) enter key is pressed 3) client clicks elsewhere in the document.
Our design decision to lock edits by word is so that no larger portion of document will be locked at anytime. This allows collaborative editing, but also ensures that we wouldn't have to deal with multiple edits at the same time.

Before the edit object becomes part of the ServerDocument (when it's still locked and only editable to the client), it will be displayed on the ServerDocument as gray text. This is so that any other client connected to the document will see what is being edited, but also know that it is still in the process of being edited, and thus untouchable.

For example, if Client 1 is typing "edit":

Client 1
This is what they both see. Client 1 edit

Client 2
This is what they both see. Client 1 edit

Client 2 will see Client 1's edit in gray, and will be unable to touch it.

Once the edit becomes part of the ServerDocument, the text will turn to black for everyone..

Deleting words:
There are two ways to delete.
1) One character at a time, using backspace or delete. This will count as a one character edit object and will immediately be sent to the server to update the global document. Even if multiple characters are deleted in a row, they each count as individual edits.
2) Highlighting a section of text and pressing delete or backspace. This will count as one edit instead of one edit per character.

Protocol

The GUI and the server send messages to each other. Any time that any key is pressed in the GUI, the GUI sends a message to the server. The server then deals with this message in a threadsafe way and responds by updating the GUI for every client.

Essentially:

One GUI sends message to server → server deals with message → server sends message to all GUIs

This is how the protocol will work in a conceptual way. We won't know the exact messages between the server and the GUI until we write the edit class, since the GUI will send information in the form of edits to the server. The server will send information to the GUI in the form of ServerDocument objects which are supposed to update the GUI. The server sends its information to the GUI when the DocumentListener realizes that the ServerDocument has changed.

Grammar for messages when something is added:
Message::= Addition || Deletion || BigDeletion
Addition::= [[\S]*[\s]?]*
Deletion::= \b
BigDeletion::= [[\S]*[\s]?]*\b

Since we defined an edit to end whenever the cursor moves, a timeout occurs, a whitespace character is added, or a delete/backspace key is pressed, the grammar is defined as so.