In this assignment, you will combine assignment 6 and 7 to create a working chat application. To do so, you will create a new object that allows users to enter chat commands consisting of the tokens produced by your scanner of assignment 7. These commands, in turn, will manipulate the avatar collection of assignment 6. You may have to create another object or change the avatar collection to keep track of the message number. As we will not have class on Mar 24, you have one and a half week to finish the assignment. However, let this not encourage you to procrastinate – you may have to fix problems with assignment 6 and 7 to get this one working. You do not have to implement undo/redo in this assignment.

**Abstract Classes**

Do the part of the last assignment that asked you to identify abstract classes.

**User Object Constructor**

Create a new class that represents a chatting user. Let us call this class the user class and an instance of it a user object.

The constructor of this class should create a new avatar. In assignment 5, you defined two avatars. In assignment 6, you allowed one of these avatars to be put in a collection. The avatar created by a user object should be the one that can be put in your avatar collection.

The constructor takes as arguments the name of the avatar, its initial X coordinate, and an instance of the avatar collection (of assignment 6) into which the avatar can be put.

Thus, each user object has a reference to an avatar list and an avatar. We will refer to them as simply the avatar and avatar list.

**User Object Properties**

A user class should define two properties, a readonly property whose value is the avatar name passed to the constructor, and an editable String property, which represents a command entered by the user. Let us call these two properties the name and command properties. These are the only properties defined by the object. The following figure uses ObjectEditor to visualize the properties of a user object:
As always, you are free to use your own names for the properties required by an assignment.

The user object should not define a property for the avatar itself, or for the avatar collection. Thus, the ObjectEditor window for a user object will not show any graphics.

**User Object Methods**

Besides getters, setters, and possibly other methods of your choice, a user object should define the following public methods to manipulate the avatar and avatar collection: join(), leave(), move(int), message(String). The first two methods take no arguments, and the last two take a single argument of type `int` and `String`, respectively. As always, you are free to choose the names of method required by an assignment.

The join method adds the avatar to the avatar collection; and the leave method removes it from the collection. The message method adds a message shape to the chat history of the avatar that displays the String argument of the method. The constraints regarding the location of the message shape are given later. The move method increments the X coordinate of the avatar by the `int` value passed to the method.

**User Object Commands**

A user object should also process four commands that invoke the four method described above. The commands are invoked by editing the command property.

The syntax for invoking these commands is given below:

- `<Command> → <Join Command> | <Leave Command> | <Move Command> | <Message Command>`
- `<Join Command> → <join token>`
- `<Leave Command> → <leave token>`
- `<Move Command> → <move token> <number token>`
- `<Message Command> → <message token> <string token>`

Here the symbol | means “or”. Thus, a command is a join command, leave command, move command, or message command. A join (leave) command consists of the single join (leave) token. The move command consists of the move token followed by a number token; and the message command consists of a message token followed by a string token. These commands invoke the corresponding methods of the user object.

You can assume that users enter only legal tokens. However, they may enter illegal combinations of these tokens such as:
leave 50

If an illegal command is entered, print a message on the console. Otherwise process the command.

You should not change the scanner to check the command syntax, that is, to parse the tokens. This task should be done by the user object.

**Message Location Constraints**

The message sent in the chat history of an avatar should be vertically aligned, as shown in the figure below. It's up to you whether you align the left, right, or middle of each message shape. It is not possible send messages concurrently, thus the messages sent by the users have a unique order. A message should be displayed above the message sent before it, and the spacing between messages should be constant, as shown in the figure below. To determine the Y coordinate of the next message, you can change the avatar collection or create a new object. You should not use static variables.

![Figure 2 Message Alignment in Avatar Collection](image-url)
**Constraints**
As mentioned above, you cannot use static variables to determine the Y coordinate of the next message.

**Demoing the Chat and the Refresh Command**
To demo the chat, write a main method that creates and displays (using ObjectEditor) an avatar collection (Figure 2) and multiple user objects representing chatting users. Thus, your display will include multiple ObjectEditor windows. Actions made in one window do not automatically update the display of other windows. However, you can invoke View ➔ Refresh command to manually refresh a window. The new version of ObjectEditor (and maybe even the old one) may give an error on this command. If it does, let me know, try changing the version, and make sure, using println() or the debugger, that the operation is working correctly. I will try and fix the problem after I return from my trip.

**Extra Credit**
Create appropriate methods and commands in the user object to invoke additional avatar operations you have implemented such as *smile*.

Process the move and message commands in separate methods of the user class. These methods will be called by some common command processing method that gets the first command token from the scanner and, based on this token, determines which command-specific method is called.

Allow a negative number to be entered as an argument to the move command: e.g. move -50. This will require you to create a new token type and change the scanner to recognize it.

Create a special session object that provides operations to create and display user objects dynamically.

**Submission Instructions**
1. Submit a print out of your code at the start of class on the submission date together with screen shots showing your code working in various cases, and a document identifying how you support various style and functionality features.

2. Upload the assignment directory in blackboard. In general, for all assignments, you should do so by midnight of the day the assignment is due. But do not change the code after you submit it in class.