In this assignment, you will use arrays to create different variable-sized collections. You will create fairly complex logical structures, with collection embedded in other objects. In fact, the main challenge in this assignment is to create such structures.

You will use ObjectEditor to display the elements of the collections. Thus, you must follow ObjectEditor conventions for read methods of a variable-sized collection. You do not need to do so for write methods. Moreover, while you should use ObjectEditor to invoke methods to remove elements, you should not use ObjectEditor to add elements to the collection. The reason is that each of these collections consists of composite objects such as a shape and ObjectEditor does not provide a convenient way to instantiate method parameters that are composite objects. You can programmatically add elements to connections in main and other methods of your class. Make sure a collection is fully populated before you ask ObjectEditor to display it so that you don’t have to interactively add elements to it.

Besides collections, you will create other kinds of objects, which will ultimately be associated with collections.

As before, the figures below are not to be taken literally and are meant to help interpret the instructions.

**Movable Label**

In lectures, a label class and interface was presented. Use these types, possibly after modifying them, to define objects that display movable text. You can assume all labels are of fixed width and height, or you can adjust the width of the label based on the text it displays. Each letter does not take the same number of pixels, but I found that on average a letter, on my laptop, takes 8.1 pixels in the X direction and 15 pixels in the Y direction. You can try different values to see what works best.
**Movable Chat History**

Create a dynamic ordered collection of labels. The collection should provide a method to add a new label to the end of the collection and another to remove the last element added (Last in First Out). The initial positions of the label are determined by the coordinates of the added label. The code that adds elements to the collection determines the initial coordinates of the added element. The add method of the collection does not change these coordinates. In my figure, the added items are aligned and in a certain order—your code does not have to create such a display but could do so. The label collection should define methods to move all elements by the same amount.
Movable Avatars with Chat History

Create new classes and interfaces for composing the knight and guard avatars with a chat history. Define methods that move each kind of avatar with its chat history.
Marching Knights with Chat History

Create a dynamic collection in which element is a knight with a chat history. Like the chat history collection, this collection is also ordered and provides operations to add a new element to the end of the collection, and remove the last element. The collection should provide a method to add a new element to front of the collection and an operation to remove the first element. The initial positions of the elements are determined by the coordinates of the added element. As in the case of the chat history, the code that adds elements to the collection determines the coordinates of the added element. The add method of the collection does not change these coordinates. In my figure, the added items are aligned and in a certain order — your code does not have to create such a display but could do so. The collection should define methods to move all elements by the same amount.

(a) Initial knights
(b) After moving
(c) After removing earliest element
Create an object that simulates a scene with knights. This object is like the scene without knights mentioned in the extra credit part of the previous assignment except that (a) the guard is now associated with a chat history, and (b) the scene has two instances of the collection modeling marching knights with chat history. One collection models the knights who have not crossed the bridge and the other the ones who have. Their exact positions do not matter at this point, but you should place them on the two sides of the bridge for future assignments. Be sure to read the extra credit feature in the previous assignment to understand what parts of the figure below are required and what are optional.
**Constraints**
As always, try and follow all style principles you have learned so far in class, and use only the concepts seen in class so far, and specified in earlier assignments. *This means you cannot use the class Object in the definition of your collection classes.* You must carefully choose the type of the element of each collection so that only legal objects in the collection can be stored in it.

**Refreshing ObjectEditor Window**
ObjectEditor may not refresh your display correctly after an update. To determine if it is ObjectEditor’s fault, execute View→Refresh. If that updates the display, let us know, and we will try to find out why automatic refresh does not work.

If manual refresh also does not update the display correctly, execute the execute the Customize→New Editor command. This command creates a new display of the object from scratch, and should reflect the current state. Again, let us know if it updates the display correctly, and we will try to determine why both manual and automatic refresh do not work.

If the new display is also not correct, show the tree view, and see if it updates correctly. If it does, let us know, as that means it is ObjectEditor’s fault. If it does not, try and debug your program to verify if it is correctly updating the displayed objects. If you are convinced your program is working correctly, let us know.