

Comp 401 – Assignment 6: Variable-Size Complex Collections

Date Assigned: Thu Feb 19, 2009

(Early) Completion Date: Thu Feb (24) 26, 2009

In this assignment, you will use arrays to create two different variable-sized collections. One of the collections will be embedded in a bigger object, which in turn will be an element of the second collection. Thus, you will create a fairly complex logical structure. In fact, the main challenge in this assignment is to create this structure.

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. However, you will not (and probably should not) use `ObjectEditor` to change the collection - the reason is that each of these collections consists of composite objects such as a shape or a collection and `ObjectEditor` does not provide a convenient way to instantiate method parameters that are composite objects. This means you do not have to follow `ObjectEditor` conventions for defining write methods of collections. You can instantiate elements of the collections and add them to the collections in main methods. Make sure a collection is fully populated before you ask `ObjectEditor` to display it.

In this and future assignments, some classes may be accidentally recognized as implementations of atomic shapes. Some of the composite graphics objects you will create such as an avatar may end up having graphics properties such as Cartesian coordinates. If the names of the classes (and super-classes) and interfaces (and super-interfaces) of these objects contain strings such as “Point” and “Oval” that denote atomic shapes, these composite graphics objects will be treated by `ObjectEditor` as atomic graphics objects. Thus, you might have to play around with the names of classes and interfaces to get the correct display. When you study annotations, you will see ways around this problem.

Later you will write methods that select certain shapes – to support this feature, you will add, in this assignment, facilities to specify the color of a shape. You can use this property to give various shapes colors of your choice.

You are required to use inheritance to a limited extent in this assignment.

Color

To each of the atomic shape classes you implemented in the previous assignment or implement in this and future assignments, add a property named “Color” of type `java.awt.Color`. `ObjectEditor` uses the value of this property to color the shape. Set this property to some color constant defined by `java.awt.Color`. Figure 1 (a) and (b) shows what happens when this

property is assigned `java.awt.Color.BLUE` and `java.awt.Color.MAGENTA`, respectively. (If you want the color to fill a shape, define a Boolean property named “Filled” and set it to true. This is not required.)



Figure 1 Coloring Shapes

Message History

Define a variable-sized collection whose elements represent message shapes. A message shape, like the name of an avatar in the previous assignment, is either a textbox or a text-label. Let us call the interface defining this collection as `M` (naturally you should use a better name for it and other interfaces named in this assignment). Figure 2 shows the ObjectEditor display of two different instances of `M` with different number of elements.

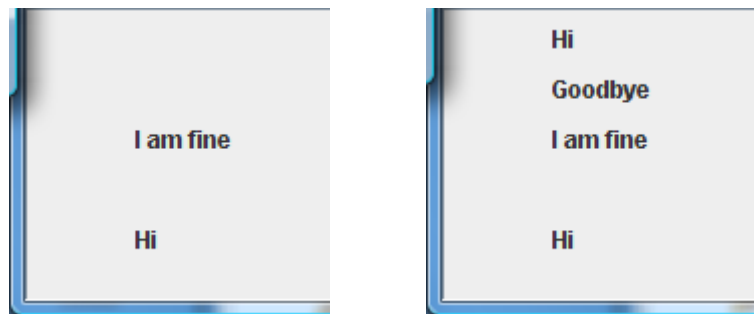


Figure 2 Two Instances with different number of message shapes

This collection will be used to implement history of chat messages. If you wish, you can ensure in the implementation of `M` that the messages are appropriately sized and placed. However, you are not required to do so – you can leave this detail to the users of `M`. In fact, in this assignment, in all test cases, *you can assume that the main method determines the placement and size of the messages*. Where you do the placement and sizing will influence the nature of the add methods you define. Remember, you do not have to follow ObjectEditor conventions for defining write methods of collections and can use main methods to compose these collections.

Communicating Avatar

In the previous assignment, you implemented two different avatars – an avatar with an atomic shape and another with a composite shape. Let call the type (interface) of the latter as `A`. Use inheritance to create an extension of `A` that has a property of type `M` to represent the messages sent by the avatar. The positions of these messages should be fixed relative to the other shapes comprising the avatar when it moves, as shown in Figure 3. Let us call the type of this extension `C`.

Collection of Communicating Avatars

Define a variable-sized collection of elements of type C. It should allow addition and removal of elements - thus, unlike M, it should not be a history. Figure 4 shows the ObjectEditor display of an instance of this collection containing two elements of type C. As before, you can use main methods to add elements to the collection.

Extra Credit

1. For each of the collections, define a method that returns an iterator that can be used to iterate through the collection.
2. Add features to these objects that you think are interesting.

Constraints

As always, try and follow all style principles you have learned. You are required to use inheritance in the definition and implementation of C. In later assignments, you will be required to use inheritance to a greater extent. You must use arrays to implement the collections – you cannot use a predefined collection type such as Vector or ArrayList.

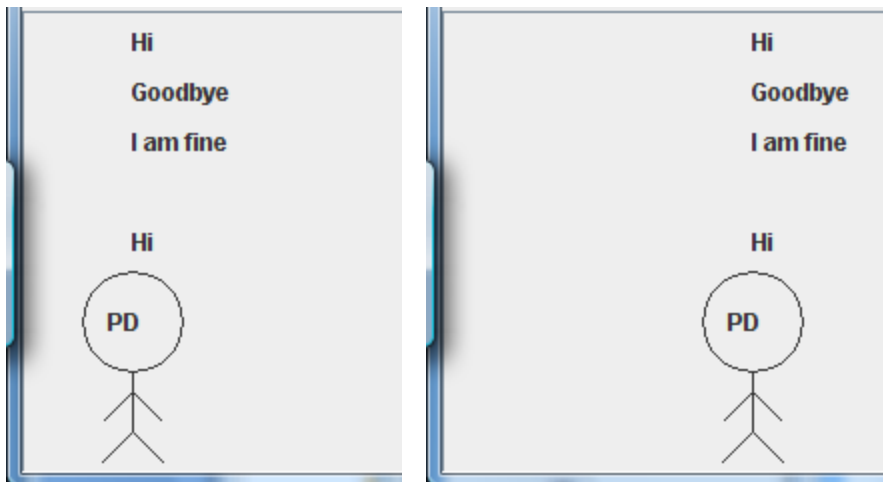


Figure 3 A moving communicating avatar

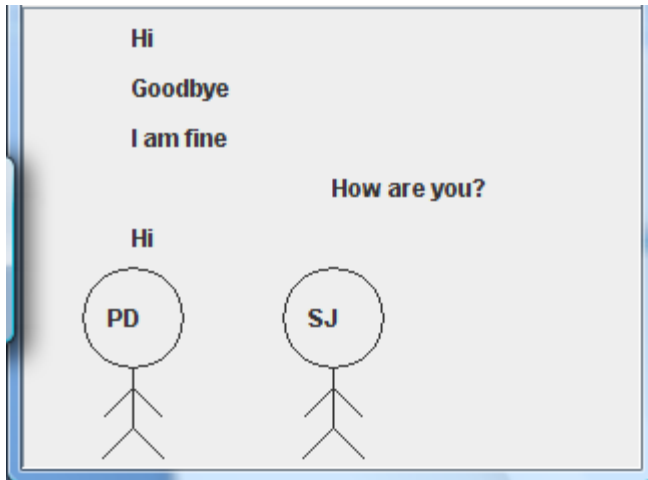


Figure 4 A variable-sized collection of communicating avatars

Submission Instructions

1. Submit a print out of your code at the start of class on the (early) submission date together with screen shots showing your code working in various cases, and a document identifying how you support regular and extra credit style 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.