

# Comp 401 – Assignment 10: Delegation, Generics, Animation, Command Objects

---

**Date Assigned: Tue April 7, 2009**

**(Early) Completion Date: (Tue April 14) Thu April 16, 2009**

In this assignment, you will apply several concepts learned in class: delegation, generics, animation, and command objects. The number of concepts is relatively large because these are mainly patterns that you can apply like a formula - there is not much variation between what you do here and the examples presented in class. As usual, extra credit covers the material that requires more thought/work.

Some of these tasks require you to replace some existing type E with a new type N in the project. The best way to do so is to give the new type the same name and put it in a new package. For example, if *AMessageHistory* is in package *collections*, give the new version of it the same name and put it in the new package *generics*. Now you can use the new version of it in the rest of the project by just changing the import. One warning, Eclipse does not show the full name of a class in a window editing it, so you may get confused as to which version you are editing.

## ***Converting an Inheriting Class to a Delegating Class***

In assignment 7, you created a new scanner class, which iterated both token objects and strings. This class was a subclass of the scanner class of assignment 4, which iterated strings. Create a new scanner class that is like the scanner class of assignment 7 except (a) it iterates only token objects, and (b) delegates to the scanner of assignment 4 instead of inheriting from it. Thus, the interface of this scanner class will be a bit different from the interface of the scanner of assignment 7 as it will have one less method. However, this should not make a difference to your parser as presumably you did not use the string iteration method in it. Show that this delegating scanner works by using it in the user-object.

## ***Generic Collections***

In assignment 6, you created two types of collections: a message history and an avatar collection. Create new versions of them that use generics to reduce the code duplication in them. This means you will have to create a common generic superclass of the two collection classes that takes type parameters and implements operations that are common to them. Similarly you should create a new generic supertype of the collection interfaces. Demonstrate that these collection classes work by using them in the rest of the project. You should still use arrays to implement the collections classes and not, for example, use ArrayList or Vector.

## ***Generic Iterators***

Change the interfaces of the scanner of assignment 4 (the delegate) and the scanner of this assignment (the delegator) so that they are subtypes/elaborations of the generic predefined iterator interface: `java.util.Iterator`. This means they should not define any methods – instead they should simply give values to the type parameter of `java.util.Iterator`.

## ***Animation, command objects and threads***

Support a new command that can be called from a user-object to animate the move of at least one of the avatars (you can define a stub method for the other avatar in case you cannot simply reuse the animation code for the two avatars). Animating a move means allowing the user to see intermediate locations of an avatar (and the associated chat history) as it moves to its new location. Thus, if the animate move command asks the avatar to move by 100, say from location 0 to 100, the avatar should pause at some locations between 0 and 100 before it rests at 100. How many intermediate locations it pauses at and how long it pauses at each intermediate location is up to the avatar class. The avatar should pause at at least one intermediate position and for at least 100 milliseconds so that the animation is clearly visible.

As *extra credit*, define editable properties in the avatar class that allows `ObjectEditor` or some other class to determine the number of intermediate locations (or distance travelled before pausing) and the pause time at each location. As in the previous assignment, you can use `ArrayList/Vector` for notifications. If the previous assignment was designed well, you should not have to write new notification code.

## ***Undo***

Allow the undo and redo command to be called from a user-object. The undo command should undo the execution of the last move, message, join and leave command and can do nothing for the other commands. The redo command can be executed only after an undo, and it redoes the command undone by the undo command. After an undo/redo command is executed, another undo/redo command cannot be executed. So your undoer has to keep track of only one user command.

As *extra credit*, allow a history based undo/redo that mimics the behavior of Eclipse, Word and other tools, allowing you to execute a series of undo and redo commands.

## ***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.