CS 474: Object Oriented Programming Languages and Environments

Fall 2014

First Smallalk project

Due time: 9:00 pm on Wednesday 10/1/2014

In this project you will define an abstract superclass that declares a common interface for a family of subclasses that manage sets and operations on sets. The abstract superclass will defer implementation of the sets to its concrete suclasses, which will use different set implementations. You are specifically responsible for implementing the abstract superclass, namely SetManager, and two concrete subclasses, namely ListSetManager and OCSetManager, in Cincom Smalltalk. The two concrete subclasses implement sets as linked lists and ordered collections, respectively. No duplicate values will be allowed in your sets, regardless of the implementation. The sets contain arbitrary objects; however, you must use the logical equivalence test to decide whether two objects are the same for the purpose of inclusion in a set. In other words, do not include an object in a set if it is logically equivalent to an object already in the set.

In your code you are not allowed to use the predefined Smalltalk class Set and its subclasses; however, you are allowed to use predefined classes LinkedList and OrderedCollection.

In addition to the above classes, you are required to include an additional class called SetApplication in your code. The main responsibility of this class is to keep a reference to an appropriate ListSetManager or OCSetManager instance. The reference will be held in an instance variable called *manager* defined in class SetApplication. There will be a single instance of SetApplication running at all times; however, a brand new the SetManager instance will be created whenever the methods startListManager and startOCManager are executed. (The actual type of SetManager created will depend on which of the two methods is actually executed.)

Abstract superclass SetManager defines two variables *X* and *Y* to hold two sets. The two concrete subclasses will bind either two linked lists or two ordered collections to *X* and *Y*.

Your submission will be graded according to the following criteria: (1) compliance with the specification below, (2) the presence of abundant code comments, (3) good code reuse (e.g., by inheriting methods instead of duplicating in multiple subclasses), and (4) conciseness (e.g., avoiding definition of unnecessary data structures). (Note: Efficiency of execution is not a criterion, but make sure that your program is reasonably responsive.)

The SetApplication and SetManager support the functionality (methods) below. In most cases, the methods defined in class SetApplication will *delegate* the implementation of the method to class *SetManager*. Both classes should have the methods listed below. (Make sure that the method name is exactly as indicated below, including capitalization.)

1. startListManager — A new instance of ListSetManager is created and bound to the *manager* variable of the running SetApplication instance. This new ListSetManager instance creates two empty linked lists and binds them to variables *X* and *Y*. Both sets will be initially empty.

2. startOCManager — A new instance of OCSetManager is created and bound to the *manager* variable of the running SetApplication instance. This new OCSetManager instance creates two empty ordered collections and binds them to variables *X* and *Y*. Both sets will be initially empty.

3. clearX — This unary method resets set *X* to be the empty set. No new set instances are created. This method is deferred in SetManager.

4. switch — This unary method swaps the sets associated with *X* and *Y*, meaning that *X* will receive the previous *Y* set and vice versa.

5. save — This unary method copies the *X* set into *Y*. The previous content of *Y* is lost. The content of *X* is not affected. The two sets must not share any data structures, that is, they can be modified independently of each other.

6. do: — This keyword method takes as input a one-argument block. The argument block is applied to each element in set *X* and the element is replaced by the value returned by the block on that element. This method is deferred in class SetManager.

7. add:. — This keyword method allows a user to add a new object to *X*. No action is taken if the argument object in question is already in the set. Otherwise, the object is added to the set. This method is deferred in SetManager.

8. remove: — This keyword method allows a user to remove an element from *X*. No action is taken if the object in question is not in the set. Otherwise, the object passed as an argument is removed from *X*. This method is deferred.

9. member: — This keyword method returns true or false depending on whether set *X* contains the argument number.

10. at: — This keyword method takes a nonnegative integer parameter *i*. The element at position *i* in the set in returned.This method is deferred.

11. member: — This keyword method returns true or false depending on whether set *X* contains the argument number.

12. union — This unary method computes that set union of sets *X* and *Y*. The result is stored as set *X*. The previous content of *X* is lost. *Y* is not modified by this operation.

13. display — This unary method displays the content of sets *X* and *Y* in the Transcript window.

You must work alone on this project. Your project code should be in a special package called CS474. Save all your code by filing out that package in the file xxx.st, where xxx denotes your last name. Submit the file using the submit link in the assignment page of the Blackboard course web site. No late submissions will be accepted.