CS 474: Object Oriented Programming Languages and Environments
Fall 2018

First Smallalk project

Due time: 7:00 pm on Saturday 10/6/2018

You are to create a *Set Calculator*, an application for performing simple set operations, in Cincom Smalltalk. *Set Calculator* defines a Graphical User Interface (GUI) for user interaction. The application implements two representations for sets, an array-based and a tree-based representation. Users are given the option of choosing one or the other implementation depending on the circumstances. *Set Calculator* manages two set instances called *X* and *Y*. The implementation consists of an abstract class called *DoubleSet* and two concrete subclasses, namely *BranchingSet* and *ArrayedSet*. Subclass *ArrayedSet* implements a set as a Smalltalk *OrderedCollection*, whereas class *BranchingSet* implements a set as a Binary Search Tree (BST). Both sets *X* and *Y* can be either a *BranchingSet* or an *ArrayedSet*; however, it is possible for *X* and *Y* to be instances of different subclasses.

Recall that the root and every internal node in a BST has two children, subject to the property that a node *n*s left child holds a value less than *n*s value, and the right child holds a value greater than *n*s value. You are not required to keep the tree balanced or to enforce the property that every internal node have two children.

No duplicate values will be allowed in your sets, regardless of the implementation. The sets contain only numerical values; however, you must use Smalltalks logical equivalence operator (as opposed to the physical identity) to decide whether two numbers are the same for the purpose of inclusion in a set. In other words, do not include a number in a set if it is logically equivalent to a number 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 class *OrderedCollection* when implementing your *ArrayedSet* class.

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 the two 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.)

When designing your user interface, make sure to include appropriate widgets (i.e., interface elements) supporting the following functionality.

1. New set creation — Use two buttons to create a new set either as an *ArrayedSet* or a *BranchingSet*. The values stored in the set are obtained by reading the content of the *Set values* field below. The new set is stored as *X*. The old value of *X* is lost forever.

2. Set values — This field allows a user to type a space-separated list of numerical values (either integer or floating point numbers). These values are used to populate a set, e.g., using the set creation widget above.

3. Mapping set elements — This widget allows an interactive user to type a block with one argument in an appropriate widget. The block is applied to all elements of set *X*. The result of the last block execution is shown in an appropriate field in the GUI display.

4. Switch sets — When selected, this widget swaps the sets associated with *X* and *Y*, meaning that *X* will receive the previous content of *Y* set and vice versa. The types of the two sets being swapped are not modified. Thus, if *X* was a *BranchingSet* before the switch, then *Y* will be a *BranchingSet* after the switch.

5. Set union — When selected, this widget performs the set union of the numbers in *X* and *Y*. The result is stored with *X*. The type (class) of the result set is the same as the type of the original *X* set. Recall that no

duplicates are allowed. This method must be coded in abstract superclass *DoubleSet*; subclasses *ArrayedSet* and *BranchingSet* must inherit this method instead of defining it. (Hint: Use other functionality in your app to carry out the set union.)

6. Perform set intersection — When selected, this widget performs the set intersection of the numbers in *X* and *Y*. The result is stored with *X*. The type (class) of the result set is the same as the type of the original *X* set.

7. List the sets — The content of the two sets are displayed in the set value widget above. The content of the widget is cleared before it displays the contents of *X* and *Y*. Sets stored as BSTs must be traversed in in-order, resulting in an ordered listing of the elements.

8. Copy X into Y — The content of *X* is copied into *Y*. The previous content of *Y* is lost forever. The type of *Y* is the same as that of *X*; however, the two sets do not share any data structures.

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.

Hints. Create a *ReadStream* over a string to parse integer and floating point numbers contained in the *set values* field above. Use class method readFrom:, which uses the stream as an argument, in order to read numeric values from the string. Use such methods as *skipThroughAll:* in order to skip over white space characters in the string.This method is defined in class Object.