STEPHEN F. AUSTIN STATE UNIVERSITY

CSC 214

LAB 4

SAMUEL JENTSCH
ACCOUNT NUMBER: CS214114

Java- Set Operations

```
1   /*
2   PROGRAM NAME: Lab 3, Sets
3   PROGRAMMER:   Samuel Jentsch
4   CLASS:        CSC 333, Spring 2014
5   INSTRUCTOR:   Dr. Strader
6   DATE STARTED: February 22, 2014
7   DUE DATE:     February 24, 2014

8   PROGRAM PURPOSE:
9   a. This program reads two sets (containing integers) from a file.
10  b. The program shows both sets, the union of the sets, intersection of
11          the sets, difference between the sets, and the symmetric set
12          difference for the two sets.

13  FILES USED:
14    lab3.dat - a file two sets.
15  ---------------------------------------------------------------------
16  */

17  import java.util.*;
18  import java.io.*;

19  public class Lab3 {

20          public static void main(String[] args) {
21                  //To use data file from instr folder.
22                  //String fileString = "../instr/lab3.dat";
23
24                  //To use with data file submitted with program.
25                  String fileString = "lab3.dat";
26
27                  loadSetsFromFile(fileString);
28          }
29
30          public static void loadSetsFromFile(String fileName) {
31                  //-------------------------------------------------------
32                  //Loads two sets from the file name provided. Expects
33                  //a file in the format:
34                  //Set1_Label Num_Elements
35                  //Element_1
36                  //Element_2
37                  //Element_3
38                  //...
39                  //Element_Num_Elements
40                  //Set2_Label Num_Elements
41                  //Element_1
42                  //Element_2
43                  //Element_3
44                  //...
45                  //Element_Num_Elements
46                  //Precondition: Data file matching format specified
47                  //provided by calling method.
48                  //Postcondition: Two sets are created as specified by the
49                  //data file and testSets is called with the two sets to
50                  //test each set operation and output the results to the
51                  //console.
52                  //-------------------------------------------------------
53
54                  //Sets being created.
55                  TreeSet<Integer> s1 = new TreeSet<Integer>();
56                  TreeSet<Integer> s2 = new TreeSet<Integer>();
57
```

```
58                      //Array to hold the sets for easier access during data insertion.
59                      ArrayList<TreeSet<Integer>> setHolder = new ArrayList<TreeSet<Integer>>();
60                      setHolder.add(s1);
61                      setHolder.add(s2);
62
63                      try {
64                              Scanner fileReader = new Scanner(new File(fileName));
65
66                              String input;
67                              String[] splitInput;
68
69                              String setName1 = "";
70                              String setName2 = "";
71                              Integer setElement;
72                              Integer setLength;
73
74                              int setNumber = 0;
75
76                              while(fileReader.hasNext()) {
77                                      //read from file
78                                      input = fileReader.nextLine();
79                                      splitInput = input.split(" ");
80
81                                      if(splitInput.length > 1) {
82                                              //New set encountered (SET_NAME #ELEMENTS)
83                                              if(setNumber == 0) {
84                                                      setName1 = splitInput[0];
85                                              } else {
86                                                      setName2 = splitInput[0];
87                                              }
88
89                                              //Get length for the set
90                                              setLength = Integer.parseInt(splitInput[1]);

91                                              for(int i = 0; i < setLength; i++) {
92                                                      //Load the number of elements in the set into the
93                                                      //set.
94                                                      setElement = Integer.parseInt(fileReader.next());
95                                                      setHolder.get(setNumber).add(setElement);
96                                              }//end for (add set elements)
97
98                                              //Fill next set
99                                              setNumber = 1;
100                                     }
101
102                             }//end while(fileReader.hasNext())
103
104                             //Test the sets loaded with set methods:
105                             testSets(setName1, s1, setName2, s2);
106
107                     } catch (FileNotFoundException e) {
108                             //File not found.
109                             e.printStackTrace();
110                     } catch (Exception ex) {
111                             //Error message if an error is encountered while processing the file.
112                             System.out.println("There was an error while loading the sets from the file.");
113                             System.out.println("Please make sure the file is in the format:\n SET_LABEL1 NUMBER_OF_ELEMENTS");
114                             System.out.println("Followed by the integer members of the first set.\nElement_1\nElement_2\n...
\nElement_NUMBER_OF_ELEMENTS");
115                             System.out.println(" SET_LABEL2 NUMBER_OF_ELEMENTS");
116                             System.out.println("Followed by the integer members of the second set.\nElement_1\nElement_2\n...
\nElement_NUMBER_OF_ELEMENTS");
117                             System.out.println("\nEach operation on the sets will be printed after the sets are \nloaded from the file.");
118                     }
```

```
119
120             }//end loadFile

121             public static void testSets(String label1, TreeSet<Integer>s1, String label2, TreeSet<Integer> s2) {
122                     //-------------------------------------------------------
123                     //Each set operation supported by the program is called
124                     //and the results are printed to the console.
125                     //Precondition: label1 (name of first set), s1 (the first
126                     //set), label2 (name of second set), s2 (second set)
127                     //passed as parameters.
128                     //Postcondition: Each set operation is invoked with s1
129                     //and s2 and the result of each operation is printed to
130                     //the console.
131                     //-------------------------------------------------------
132                     System.out.printf("%s: %s\n", label1, s1.toString());
133                     System.out.printf("%s: %s\n", label2, s2.toString());
134                     System.out.printf("Set Union: %s\n", set_union(s1, s2).toString());
135                     System.out.printf("Set Intersection: %s\n", set_intersection(s1, s2).toString());
136                     System.out.printf("Set Difference (%s - %s): %s\n", label1, label2, set_difference(s1, s2).toString());
137                     System.out.printf("Set Difference (%s - %s): %s\n", label2, label1, set_difference(s2, s1).toString());
138                     System.out.printf("Symmetric Set Difference: %s\n", set_symmetric_difference(s1, s2));
139             }
140
141             public static TreeSet<Integer> set_union(TreeSet<Integer> s1, TreeSet<Integer> s2) {
142                     //-------------------------------------------------------
143                     //Union: items in either s1 or items in s2.
144                     //The union of s1 and s2 is set to a new set union and
145                     //returned.
146                     //Precondition:s1 and s2 (TreeSets) passed as parameters.
147                     //Postcondition: The union of s1 and s2 is returned.
148                     //union should contain all elements in s1 and all elements
149                     //in s2.
150                     //-------------------------------------------------------
151
152                     TreeSet<Integer> union = new TreeSet<Integer>();
153
154                     union.addAll(s1);
155                     union.addAll(s2);
156
157                     return union;
158             }
159
160             public static TreeSet<Integer> set_intersection(TreeSet<Integer> s1, TreeSet<Integer> s2) {
161                     //-------------------------------------------------------
162                     //Intersection: Items in both s1 AND s2.
163                     //The difference between s1 and s2 is calculated by taking
164                     //the union of s1 and s2, and removing the difference
165                     //(s1 - s2) and (s2 - s1), effectively removing all
166                     //elements present in s1 but not in s2, and all elements
167                     //in s2 but not in s1 respectively.
168                     //Precondition: s1 and s2 (TreeSets) passed as parameters.
169                     //Postcondition: A new set containing the intersection of
170                     //s1 and s2 is created and returned.
171                     //-------------------------------------------------------
172
173                     TreeSet<Integer> intersection = new TreeSet<Integer>();
174                     TreeSet<Integer> difference1 = new TreeSet<Integer>();
175                     TreeSet<Integer> difference2 = new TreeSet<Integer>();

176                     difference1 = set_difference(s1, s2);
177                     //difference1 is now s1 - s2 (items in s1 not in s2).
178
179                     difference2 = set_difference(s2, s1);
180                     //difference2 is now s2 - s1 (items in s2 not in s1).
```

```
181
182
183                     intersection = set_union(s1, s2);
184                     intersection.removeAll(difference1);
185                     intersection.removeAll(difference2);
186
187                     return intersection;
188             }
189
190             public static TreeSet<Integer> set_difference(TreeSet<Integer> s1, TreeSet<Integer> s2) {
191                     //-----------------------------------------------------------
192                     //Difference: s1 - s2. Items in s2 removed from the items
193                     //in s1.
194                     //Precondition: s1 and s2 (TreeSets) passed as parameters.
195                     //Postcondition: A new set containing the difference
196                     //(s1 - s2), or items present in s1 that are not present
197                     //in s2, is created and returned.
198                     //-----------------------------------------------------------
199
200                     TreeSet<Integer> difference = new TreeSet<Integer>();
201                     difference.addAll(s1);
202                     difference.removeAll(s2);
203
204                     return difference;
205             }
206
207             public static TreeSet<Integer> set_symmetric_difference(TreeSet<Integer> s1, TreeSet<Integer> s2) {
208                     //-----------------------------------------------------------
209                     //Symmetric difference: union(s1,s2) - intersection(s1,s2)
210                     //Precondition: s1 and s2 (TreeSets) passed as parameters.
211                     //Postcondition: A new set containing the symmetric
212                     //difference between s1 and s2 is created and returned.
213                     //-----------------------------------------------------------
214
215                     TreeSet<Integer> symmetricDifference = new TreeSet<Integer>();
216                     //Get union of s1 and s2
217                     symmetricDifference.addAll(set_union(s1, s2));
218                     //Remove the intersection of s1 and s2, yielding symmetric difference.
219                     symmetricDifference.removeAll(set_intersection(s1, s2));
220
221                     return symmetricDifference;
222             }
223
224     }
```

Test Input (File)

```
A 5
1
2
3
5
4
B 3
3
7
12
```

Test Run for Java Program (Output)

```
A: [1, 2, 3, 4, 5]
B: [3, 7, 12]
Set Union: [1, 2, 3, 4, 5, 7, 12]
Set Intersection: [3]
Set Difference (A - B): [1, 2, 4, 5]
Set Difference (B - A): [7, 12]
Symmetric Set Difference: [1, 2, 4, 5, 7, 12]
```

Prolog (Intersection)

```
1   /*
2   ---------------------------------------------------------------------
3   PROGRAM NAME: Lab 3, Prolog Problem
4   PROGRAMMER:   Samuel Jentsch
5   CLASS:        CSC 333, Spring 2014
6   INSTRUCTOR:   Dr. Strader
7   DATE STARTED: February 23, 2014
8   DUE DATE:     February 25, 2014
9   REFERENCES:   Dr. Strader: assignment information sheet
10  ---------------------------------------------------------------------
11  */

12  /*From the Prolog Class Handout*/
13  member(X, [X|Tail).
14  member(X, [Head|Tail) :- member(X, Tail).

15  /****List Intersection*****/
16  %If the first set is empty (or the second set), there is no intersection.
17  list_intersection([],S2,[]).
18  %If H1 (member of first set) is a member of the second set S2, add it to the
19  %intersection set.
20  list_intersection([H1|T1],S2,[H1|T3]) :- member(H1,S2), intersection(T1,S2,T3).
21  %If H1 (member of first set) is not a member of the second set S2, do not add
22  %it to the intersection set.
23  list_intersection([H1|T1],S2,S3) :- \+ member(H1,S2), intersection(T1,S2,S3).
```

Prolog Test Run

```
list_intersection([1,2,3,4,5], [3,7,12], I).
I = [3].
```