

O'BrienLukeAssignment5.java

```
1 import java.util.Scanner;
2 import java.util.ArrayList;
3 import java.util.Iterator;
4
5 import java.io.File;
6 import java.io.IOException;
7
8 /*
9  * Name:      Luke O'Brien
10 * Class:     Data Structure and Algorithms
11 * Section:   002
12 * Assignment: Assignmnet-5
13 * Due Date:  26 Feb 2020
14 *
15 * Desc:
16 * The program takes the four files into a Scanner then parses the data.
17 * inputing it all into the "GenericStack" Data structure.
18 * The program then prints out the individual stacks before merging them
19 * then reversing them. After they are reversed, the final product is
20 * it then printed out.
21 *
22 * after both types of stacks have gone through that process, the final
23 * stacks for each type is printed out in side by side format using the
24 * method printTwoStacks
25 *
26 * Added methods to make my life easier:
27 * printHeader(String);
28 * iterator(); //Under GenericStack in order to use iterator in the program
29 *
30 * ISSUES:
31 * I could not for the life of me, sort the last two digits in the while
32 * loop inside mergeStacks(). I understand it's because of where my next()
33 * is, but if I were to move it, the rest of the method would break.
34 */
35
36 public class OBrienLukeAssignment5
37 {
38     public static void main(String[] args) throws IOException
39     {
40         Scanner intFileOne = new Scanner(new File("numbers1.txt"));
41         Scanner intFileTwo = new Scanner(new File("numbers2.txt"));
42
43         Scanner strFileOne = new Scanner(new File("mountains1.txt"));
44         Scanner strFileTwo = new Scanner(new File("mountains2.txt"));
45
46         GenericStack<Integer> intStackOne = new GenericStack<>();
47         GenericStack<Integer> intStackTwo = new GenericStack<>();
48         GenericStack<Integer> intFinStack = new GenericStack<>();
49
50         GenericStack<String> strStackOne = new GenericStack<>();
51         GenericStack<String> strStackTwo = new GenericStack<>();
52         GenericStack<String> strFinStack = new GenericStack<>();
53
54         GenericStack<Integer> intPrintStack = new GenericStack<>();
55         GenericStack<String> strPrintStack = new GenericStack<>();
56
57     }
```

```

58     while(intFileOne.hasNextInt()) {
59         intStackOne.push(intFileOne.nextInt());
60     }
61     while(intFileTwo.hasNextInt()) {
62         intStackTwo.push(intFileTwo.nextInt());
63     }
64     while(strFileOne.hasNextLine()) {
65         strStackOne.push(strFileOne.nextLine());
66     }
67     while(strFileTwo.hasNextLine()) {
68         strStackTwo.push(strFileTwo.nextLine());
69     }
70
71     printHeader("Number Stack 1, Filled with Integers from 'number1.txt'");
72     printStack(intStackOne);
73     printHeader("Number Stack 2, Filled with Integers from 'number2.txt'");
74     printStack(intStackTwo);
75     mergeStacks(intStackOne, intStackTwo, intFinStack);
76     reverseStack(intFinStack, intPrintStack);
77     printHeader("Merged numbers stack");
78     printStack(intPrintStack);
79
80     printHeader("String Stack 1, Filled with Integers from 'montains1.txt'");
81     printStack(strStackOne);
82     printHeader("String Stack 2, Filled with Integers from 'montains2.txt'");
83     printStack(strStackTwo);
84     mergeStacks(strStackOne, strStackTwo, strFinStack);
85     reverseStack(strFinStack, strPrintStack);
86     printHeader("Merged String stack");
87     printStack(strPrintStack);
88
89     printTwoStacks(intPrintStack, strPrintStack);
90
91     intFileOne.close();
92     intFileTwo.close();
93     strFileOne.close();
94     strFileTwo.close();
95 }
96
97 public static<T> void printStack(GenericStack<T> stack)
98 {
99     Iterator<T> iterStack = stack.iterator();
100
101     while(iterStack.hasNext()) {
102         System.out.println(iterStack.next());
103     }
104 }
105 public static<T, F> void printTwoStacks(GenericStack<T> stackOne, GenericStack<F>
stackTwo)
106 {
107     int stackOneSize = stackOne.size();
108     int stackTwoSize = stackTwo.size();
109
110     Iterator<T> iterOne = stackOne.iterator();
111     Iterator<F> iterTwo = stackTwo.iterator();
112
113     System.out.println("-----");

```

```

114     System.out.printf("%-15s%s\n", "Integer", "String");
115     System.out.println("-----");
116
117     if(stackOneSize >= stackTwoSize) {
118         while(iterTwo.hasNext()) {
119             System.out.printf("%-15s%s\n", iterOne.next(), iterTwo.next());
120         }
121         while(iterOne.hasNext()) {
122             System.out.printf("%-15s%s\n", iterOne.next(), "-----");
123         }
124     }
125     else {
126         while(iterOne.hasNext()) {
127             System.out.printf("%-15s%s\n", iterOne.next(), iterTwo.next());
128         }
129         while(iterTwo.hasNext()) {
130             System.out.printf("%-15s%s\n", "-----", iterTwo.next());
131         }
132     }
133
134 }
135
136 public static<T extends Comparable<T>> void mergeStacks(GenericStack<T> stackOne,
GenericStack<T> stackTwo, GenericStack<T> stackFinal)
137 {
138     Iterator<T> iterOne = stackOne.iterator();
139     Iterator<T> iterTwo = stackTwo.iterator();
140
141     T one = iterOne.next();
142     T two = iterTwo.next();
143
144
145     while(iterOne.hasNext() && iterTwo.hasNext()) {
146         if(one.toString().compareTo(two.toString()) >=
two.toString().compareTo(one.toString())) {
147             stackFinal.push(one);
148             one = iterOne.next();
149         }
150         else if(one.toString().compareTo(two.toString()) <
two.toString().compareTo(one.toString())) {
151             stackFinal.push(two);
152             two = iterTwo.next();
153         }
154     }
155     if(one.toString().compareTo(two.toString()) >=
two.toString().compareTo(one.toString())) {
156         stackFinal.push(one);
157     }
158     else if(one.toString().compareTo(two.toString()) <
two.toString().compareTo(one.toString())){
159         stackFinal.push(two);
160     }
161     while(iterTwo.hasNext()) {
162         stackFinal.push(iterTwo.next());
163     }
164     while(iterOne.hasNext()) {
165         stackFinal.push(iterOne.next());

```

```

166     }
167 }
168 public static<T> void reverseStack(GenericStack<T> inputStack, GenericStack<T>
    outputStack)
169 {
170     while(!inputStack.isEmpty()) {
171         outputStack.push(inputStack.pop());
172     }
173 }
174
175 public static void printHeader(String str) {
176     System.out.println("\n\n*****");
177     System.out.printf("%s\n", str);
178     System.out.println("*****");
179 }
180 }
181
182 class GenericStack<E> implements Iterable<E>
183 {
184     private ArrayList<E> genStack;
185
186     GenericStack(){
187         genStack = new ArrayList<>();
188     }
189
190     public void push(E item) {
191         genStack.add(item);
192     }
193
194     public E peek() {
195         return genStack.get(genStack.size()-1);
196     }
197
198     public E pop() {
199         E temp = genStack.get(genStack.size()-1);
200         genStack.remove(genStack.size()-1);
201         return temp;
202     }
203
204     public int size() {
205         return genStack.size();
206     }
207
208     public boolean isEmpty() {
209         if(genStack.size() == 0) {
210             return true;
211         }
212         else {
213             return false;
214         }
215     }
216
217     public Iterator<E> iterator() {
218         return genStack.iterator();
219     }
220 }
221

```