323.35 Project 1: LL implementation of Stack, Queue, and Ordered List Java

Student: Matthew Flammia

Project Due date: 9/3/2020

****
IV. main ()
****
Step 0: inFile open input file from args[0]
outFile1, outFile2, outFile3 open from args[1], args[2], args[3]
Step 1: S buildStack(inFile) // Algorithm steps is given below
Step 2: dumpStack(S, outFile1)

// on your own; see the method's description in the above

Step 3: close inFile
Step 4: re-open inFile
Step 5: Q buildQueue(inFile) // Algorithm steps is given below
Step 6: dumpQueue(Q, outFile2)

// on your own; see the method's description in the above

Step 7: close inFile
Step 8: re-open inFile
Step 9: LL buildList(inFile) // Algorithm steps is given below
Step 10: printList(LL, outFile3)

// on your own; see the method's description in the above

Step 11: close all files

Source Code

```
import java.util.NoSuchElementException;
import java.util.Scanner;
import java.io.*;
import java.io.PrintStream;
/**
* Matthew Flammia, 23661371
* CSCI 355 Project 1
* To execute, have the source data file in working directory.
* args[0] is the source data file
* args[1] is the stack file name
* args[2] is the queue file name
* args[3] is the linked list file name
* Example run commmand:
* java -cp bin FlammiaM_Project1_Java StackQueueList_Data.txt outFile1.txt
outFile2.txt outFile3.txt
**/
public class FlammiaM_Project1_Java{
	public static void main(String[] args) throws Exception{
		//file io setup
		File data = new File(args[0]);
		//stack portion
		LLStack stackPart = new LLStack();
		stackPart.buildStack(data);
		stackPart.dumpStack(args[1]);
		//queue
		LLQueue queuePart = new LLQueue();
		queuePart.buildQueue(data);
		queuePart.dumpQueue(args[2]);
		//LList
		LList listPart = new LList();
		listPart.buildList(data);
		listPart.printList(args[3]);
	}
}

class listNode{
	//variables
	int data;
	listNode next;
	//constructors
	listNode(int data){
		this.data = data;
	}
}

class LLStack{
```

```
	//variables
	listNode top;
	//constructors
	LLStack(){
		this.top = new listNode(99999);
		this.top.next = null;
	}
	//methods
	void push(listNode node){
		node.next = this.top;
		this.top = node;
	}
	listNode pop(){
		//checks if empty; then returns null if is
		if(isEmpty()){
			return null;
		}
		//makes a temp node, stores the value of top, then sets new top to
top.next
		listNode temp;
		temp = this.top;
		this.top = this.top.next;
		return temp;
	}
	boolean isEmpty(){
		//checks if stack is empty
		if(this.top.next==null){
			return true;
		}
		return false;
	}
	void printTop(){
		//prints to console, or whatever outstream is set to
		System.out.println(this.top.data);
	}
	void buildStack(File data) throws Exception{
		//begins scanning input file for int
		Scanner input = new Scanner(data);
		while(input.hasNextInt()){
			//goes through all integers and adds them to stack
			listNode node = new listNode(input.nextInt());
			this.push(node);
		}
	}
	void dumpStack(String outFile1) throws IOException{
		//sets output stream to the name specified in args
		PrintStream fileOut = new PrintStream(outFile1);
		System.setOut(fileOut);
```

```
            //loops the stack, and empties it and prints it
            while(!this.isEmpty()){
                  this.printTop();
                  this.pop();
            }
            fileOut.close();
      }
}

class LLQueue{
      //variables
      listNode head;
      listNode tail;
      listNode dummy;
      //methods
      LLQueue(){
            //constructor
            dummy = new listNode(99999);
            head = new listNode(0);
            tail = new listNode(0);
            head.next = dummy;
            tail.next = dummy;
      }
      void insertQ(listNode node){
            //special case for first node insert
            if(this.isEmpty()){
                  dummy.next = node;
                  tail.next = node;
            }
            //case for all other nodes after first
            else{
                  tail.next.next = node;
                  tail.next = node;
            }
      }
      listNode deleteQ(){
            //prevents deleting empty queue
            if(isEmpty()){
                  return null;
            }
            //special case for single node in queue
            if(dummy.next == tail.next){
                  listNode temp = dummy.next;
                  tail.next = dummy;
                  dummy.next = null;
                  return temp;
            }
            //generic node removal
```

```
            else{
                  listNode temp = dummy.next;
                  dummy.next = dummy.next.next;
                  return temp;
            }
      }
      boolean isEmpty(){
            if(tail.next == dummy){
                  return true;
            }
            return false;
      }
      void buildQueue(File data) throws Exception{
            //begins scanning input file for int
            Scanner input = new Scanner(data);
            while(input.hasNextInt()){
                  //goes through all integers and adds them to queue
                  listNode node = new listNode(input.nextInt());
                  this.insertQ(node);
            }
      }
      void dumpQueue(String outFile2) throws IOException{
            //creates file output stream
            PrintStream fileOut = new PrintStream(outFile2);
            System.setOut(fileOut);
            //loops until empty, printing and removing nodes
            while(!this.isEmpty()){
                  listNode temp = this.deleteQ();
                  System.out.println(temp.data);
            }
            fileOut.close();
      }
}

class LList{
      //variables
      listNode head;
      listNode dummy;
      //methods
      LList(){
            //constructor
            head = new listNode(0);
            dummy = new listNode(99999);
            head.next = dummy;
      }
      listNode findSpot(listNode node){
            listNode spot = dummy;
            //loops until spot reaches the final node
```

```
		while(spot.next != null){
			if(spot.next.data >= node.data){
				//if spots data is greater, get ready for insert
				return spot;
			}
			spot = spot.next;
		}
		//if spot reaches the end, return the final position
		return spot;
	}
	void insertOneNode(listNode spot, listNode node){
		//simple insert code
		node.next = spot.next;
		spot.next = node;
	}
	void buildList(File data) throws Exception{
		//begins scanning input file for int
		Scanner input = new Scanner(data);
		while(input.hasNextInt()){
			//goes through all integers and adds them to list
			listNode node = new listNode(input.nextInt());
			this.insertOneNode(this.findSpot(node), node);
		}
	}
	void printList(String outFile3) throws IOException{
		//creates file stream
		PrintStream fileOut = new PrintStream(outFile3);
		System.setOut(fileOut);
		//creates read head to traverse
		listNode readHead = this.head.next;
		//initial printout
		System.out.print("listHead-->");
		while(readHead.next != null){
			//read head loops, printing the data value of itself and its
next
			System.out.print("("+readHead.data+",
"+readHead.next.data+")-->");
			readHead = readHead.next;
		}
		//prints final data message
		System.out.println("("+readHead.data+", NULL)-->NULL");
		fileOut.close();
	}
}
```

Output

outFile1 -Stack output

637
361
35
834
935
32
538
420
19
818
945
9599
3999
588
95
702
16
739
322
91
388
213
2255
637
361
730

outFile2 -Queue output

730
361
637
2255
213
388
91
322
739
16
702
95
588
3999
9599
945
818

19
420
538
32
935
834
35
361
637

outFile3 -List output

listHead-->(99999, 16)-->(16, 19)-->(19, 32)-->(32, 35)-->(35, 91)-->(91, 95)-->(95, 213)-->(213, 322)-->(322, 361)-->(361, 361)-->(361, 388)-->(388, 420)-->(420, 538)-->(538, 588)-->(588, 637)-->(637, 637)-->(637, 702)-->(702, 730)-->(730, 739)-->(739, 818)-->(818, 834)-->(834, 935)-->(935, 945)-->(945, 2255)-->(2255, 3999)-->(3999, 9599)-->(9599, NULL)-->NULL