CS 1150 Design Notebook Required Sections

Step 1: Problem Statement

This assignment will first use Java's built in stack to change only certain values. The next part is to create a generic stack class to take in any type of Stack/Arraylist. This class will make an arraylist act like a stack using public methods. Then fill the stacks with given files and sort and merge the two stacks and display each step.

Step 2: Understandings

- What I Know:
  - Objects
  - Arrays
  - Interfaces
  -
- What I Don't Know:
  - Stacks, little confusing to work with

Step 3: Pseudocode

- Main: Part One
  - Create given array
  - Create Stack
    - Stack<Integer> name = new Stack<>()
  - Fill Stack, use for loop
  - Create a temp Stack
    - Want Values to be opposite of first stack
    - Use For Loop
  - Call method *replaceZerosWithTen*
    - replaceZerosWithTen(Stack<Integer> stack)
  - Call printStack Method
- Main: Part Two
  - Create 2 GenericStack variables
    - GenericStack<Integer> stackNameOne = new GenericStack<>();
    - GenericStack<Integer> stackNameTwo = new GenericStack<>();
  - Take in values from given files to fill array
  - Call fillStack Method
    - fillStack(stackName, readFileName);
  - Display both stacks using generic method
  - Sort both stacks using generic method
    - sortStack(stackName);
  - Create new GenericStack to hold both stacks
    - GenericStack<Integer> mergedStack = new GenericStack>();
  - Use generic merge method to combine stacks in order
  - Display merged Stack
  - Repeat for two string stacks
  -

- SortStack(GenericStack<E> stack) {
  - Takes in GenericStack Variable, type E
  - Implements Comparable

- Create an new GenericStack to hold value largest to lowest (low on top)
- Use stack to sort newStack
- Use compareTo to sort values, want smallest value
- Use newStack to fill stack with lowest to largest (largest on top)

Step 4: Lesson Learned

It took me a while to figure out the sorting and merging methods. I knew I wanted a stack with the smallest value on top, so my original stack had the largest on top, but it was confusing how I made that happen. In the merge method I realized I made more code than I needed because it remade the sortStack method again with extra conditions, I realized I could just remove the sorting code and just compare what value is bigger and add that one first. I also forgot that making a new stack equal my original stack so, GenericStack<E> newStack = stack, effects both stacks when I add or remove values so I had to change my printing and filing code so stack didn't change.

Step 5: Code

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.ArrayList;
import java.util.Scanner;
import java.util.Stack;

/*
Isaiah Hoffer
CS1450 (M/W)
3/5/25
Assignment 5
Create Generic Stacks and fill them with String or Integer Values and sort them
using compareTo and merge them together in order of lowest to highest (lowest on top).
Create a GenericStack Class to take in Generic type E (String or Integer) and create an
arrayList to ask like a Stack, create public methods to get stack like behavior.
*/


public class HofferIsaiahAssignment5 {

	public static void main(String[] args) throws FileNotFoundException {


	/**********************************************************************
		*                                                    PART ONE
**********************************************************************
		*/

		//Creating Array
		int[] numberArray = {0, 0, 4, 3, 0, 0, 2, 1, 0, 0};
```

```
        //Creating Stack
        Stack<Integer> integerStack = new Stack<>();

        //Filling Array
        for(int i = 0; i < numberArray.length; i++) {

                integerStack.push(numberArray[i]);

        }//For


        //Calling Method To Make 0's To 10's
        replaceZerosWithTen(integerStack);

        //Pretext
        System.out.printf("-------------------------------------\n"
                        + "\t    Part One:\t\t\n"
                        + "-------------------------------------\n");

        //Displaying Stack
        printStack(integerStack);



    /**************************************************************************
             *                                                        PART TWO

**************************************************************************
             */

        //Pretext
        System.out.printf("-----------------------------------\n"
                        + "\t    Part Two:\t\t\n"
                        + "-----------------------------------\n");



    /*************************************************************
                         *                                                INTEGERS

*************************************************************
                         */
        //Creating Generic Stacks
        GenericStack<Integer> integerStackOne = new GenericStack<>();
        GenericStack<Integer> integerStackTwo = new GenericStack<>();

        //Creating Files
        File integerOneFile = new File("integers1.txt");
        File integerTwoFile = new File("integers2.txt");
```

```
//Reading Files
Scanner readIntegerOneFile = new Scanner(integerOneFile);
Scanner readIntegerTwoFile = new Scanner(integerTwoFile);


//Filling Integer Stacks
fillIntStack(integerStackOne, readIntegerOneFile);
fillIntStack(integerStackTwo, readIntegerTwoFile);

//Printing Stacks/Arrays
//Pretext
System.out.printf("-------------------------------------\n"
                  + "\tInteger Stack One:\t\t\n"
                  + "-------------------------------------\n");

printStack(integerStackOne);

//Pretext
System.out.printf("-------------------------------------\n"
                  + "\tInteger Stack Two:\t\t\n"
                  + "-------------------------------------\n");

printStack(integerStackTwo);

//Sorting Stacks
sortStack(integerStackOne);
sortStack(integerStackTwo);

//Pretext
System.out.printf("-------------------------------------\n"
                  + "     Integer Stack One Sorted:\t\n"
                  + "-------------------------------------\n");

printStack(integerStackOne); //Displays Stack

//Pretext
System.out.printf("-------------------------------------\n"
                  + "     Integer Stack Two Sorted:\t\n"
                  + "-------------------------------------\n");

printStack(integerStackTwo); //Displays Stack

//Merging Both Stacks
GenericStack<Integer> mergedIntegerStack = mergeStacks(
                                    integerStackOne, integerStackTwo);


//Pretext
System.out.printf("-------------------------------------\n"
                  + "     Integer Stack Merged Sorted:\t\n"
                  + "-------------------------------------\n");
```

```
        //Displaying mergedStacks
        printStack(mergedIntegerStack);

        //Closing Files
        readIntegerOneFile.close();
        readIntegerTwoFile.close();


    /************************************************************
                          *                                                    STRINGS
************************************************************
                          */

        //Creating Generic Stacks
        GenericStack<String> stringStackOne = new GenericStack<>();
        GenericStack<String> stringStackTwo = new GenericStack<>();

        //Creating Files
        File stringOneFile = new File("strings1.txt");
        File stringTwoFile = new File("strings2.txt");

        //Reading Files
        Scanner readStringOneFile = new Scanner(stringOneFile);
        Scanner readStringTwoFile = new Scanner(stringTwoFile);

        //Filling String Stacks
        fillStringStack(stringStackOne, readStringOneFile);
        fillStringStack(stringStackTwo, readStringTwoFile);


        //Printing Stacks/Arrays
        //Pretext
        System.out.printf("-----------------------------------\n"
                          + "\tString Stack One:\t\t\n"
                          + "-----------------------------------\n");

        printStack(stringStackOne);

        //Pretext
        System.out.printf("-----------------------------------\n"
                          + "\tString Stack Two:\t\t\n"
                          + "-----------------------------------\n");

        printStack(stringStackTwo);

        //Sorting Stacks
        sortStack(stringStackOne);
        sortStack(stringStackTwo);
```

```
        //Pretext
        System.out.printf("-------------------------------------\n"
                        + "     String Stack One Sorted:\t\n"
                        + "-------------------------------------\n");

        printStack(stringStackOne); //Displays Stack

        //Pretext
        System.out.printf("-------------------------------------\n"
                        + "     String Stack Two Sorted:\t\n"
                        + "-------------------------------------\n");

        printStack(stringStackTwo); //Displays Stack

        //Merging Both Stacks
        GenericStack<String> mergedStringStack = mergeStacks(
                                        stringStackOne, stringStackTwo);

        //Pretext
        System.out.printf("-------------------------------------\n"
                        + "     String Stack Merged Sorted:\t\n"
                        + "-------------------------------------\n");

        //Displaying mergedStacks
        printStack(mergedStringStack);

        //Closing Files
        readStringOneFile.close();
        readStringTwoFile.close();



}//main

//Method To Replace Any AND Only Zeros Within The Stack
public static void replaceZerosWithTen(Stack<Integer> stack) {

        //Temp Stack To Hold Values
        Stack<Integer> tempStack = new Stack<>(); //Want To Have Reversed Stack

        //Filling tempStack
        while(!stack.isEmpty()) { //Goes Tell Stack Is Empty

                tempStack.push(stack.pop()); // Makes Stack Reversed And Empties stack

//{0, 0, 1, 2, 0, 0, 3, 4, 0, 0}
        }//For

        //Loops Through Size Of Stack To Replace All Values
        while(!tempStack.isEmpty()) { //Goes Tell tempStack Is Empty
```

```
                    //Checking If Value Is 0
                    if(tempStack.peek() == 0) {

                            tempStack.pop(); //Only Remove If Value is 0
                            stack.push(10);

                            }//If

                            //Any Other Value
                            else {

                            stack.push(tempStack.pop()); //Adds Value To Stack And Removes It
From tempStack

                            }//Else
            }//For

    }//replaceZerosWithTen Method


    //Print Stack Method -- Prints Stack
    public static void printStack(Stack<Integer> stack) {

            //Create tempStack
            Stack<Integer> tempStack = new Stack<>();

            //Removes stack Values And Add Them To tempStack And Displays It
            while(!stack.isEmpty()) {

                    int value = stack.pop();

                    tempStack.push(value);

                    System.out.println(value);

            }//While

            //Filling stack To Original State
            while(!tempStack.isEmpty()) {

                    stack.push(tempStack.pop());

            }//While

    }//printStack Method

    //Generic printStack Method
    public static <E> void printStack(GenericStack<E> stack) {

            //Create tempStack
            GenericStack<E> tempStack = new GenericStack<>();
```

```
        //Removes stack Values And Add Them To tempStack And Displays It
        while(!stack.isEmpty()) {

                E value = stack.pop();

                tempStack.push(value);

                System.out.println(value);

        }//While

        //Filling stack To Original State
        while(!tempStack.isEmpty()) {

                stack.push(tempStack.pop());

        }//While

}//printStack

//Fills Integer Stack/ArrayList From Given File -- OPTIONAL
public static void fillIntStack(GenericStack<Integer> stack, Scanner readFile) {

        //Push Values To Array/Stack Until No More Values In File
        while(readFile.hasNext()) {

                stack.push(readFile.nextInt());

        }//While

}//fillIntStack --Optional

//Fills String Stack/ArrayList From Given File -- OPTIONAL
        public static void fillStringStack(GenericStack<String> stack, Scanner readFile) {

                //Push Values To Array/Stack Until No More Values In File
                while(readFile.hasNext()) {

                        stack.push(readFile.nextLine());

                }//While

        }//fillStringStack --Optional



//Generic Method To Sort Any Stack/ArrayList
public static <E extends Comparable<E>> void sortStack(GenericStack<E> stack) {

        GenericStack<E> tempStack = new GenericStack<>();
```

```
            while(!stack.isEmpty()) {

                  E value = stack.pop(); //Gets Stack Value

                  //Looping To Get Highest To Lowest Order
                  while(!tempStack.isEmpty() && tempStack.peek().compareTo(value) < 0) {
//Finds Smaller Values


                        stack.push(tempStack.pop()); //Add Smaller Values Back

                  }//While

                  tempStack.push(value); //Gets Values Is Correct Order -- Lowest On Top

            }//While

            while(!tempStack.isEmpty()) {

                  stack.push(tempStack.pop()); //Gets Stack In Correct Order

            }//While

      }//sortStack Method

      public static <E extends Comparable<E>> GenericStack <E> mergeStacks(
                  GenericStack<E> stackOne, GenericStack<E> stackTwo) {

            GenericStack<E> mergedStack = new GenericStack<>();

            while(!stackOne.isEmpty() || !stackTwo.isEmpty()) {

                  //Creating Value To Push To mergeStack
                  E value = null;

                  //No Need To Check If Array Is Empty
                  if(stackOne.isEmpty()) {

                        value = stackTwo.pop(); //Sets Value To Top Of StackTwo
                  }//If

                  //No Need To Check If Array Is Empty
                  else if(stackTwo.isEmpty()) {

                        value = stackOne.pop(); //Sets Value To Top Of StackOne
                  }//Else If

                  //Checks If Top StackOne Is Smaller Than Top StackTwo
                  else if(stackOne.peek().compareTo(stackTwo.peek()) < 0) {
```

```
                    value = stackTwo.pop();

                }//Else If

                //Checks If Top StackTwo Is Smaller Than Top StackOne
                else if(stackTwo.peek().compareTo(stackOne.peek()) < 0) {

                    value = stackOne.pop();

                }//Else If

                mergedStack.push(value); //Adds Value To New Array -- Smallest On Top


            }//While

            return mergedStack;

        }//While


}//class


class GenericStack<E> {

        //Private Data
        ArrayList<E> list;


        GenericStack() {

            list = new ArrayList<>();

        }//GenericStack Constuctor

        //Checks If ArrayList/Stack Is Empty
        public boolean isEmpty() {

            return list.isEmpty(); // Returns True Or False

        }//isEmpty Method

        //Returns Size Of List
        public int getSize() {

            return list.size(); //Returns list's Size

        }//getSize Method
```

```
        //
        public E peek() {

                E lastValue = list.get(getSize()-1);


                return lastValue; //Shows Last Value In List

        }//Peek Method

        public E pop() {

                //Value To Return
                E lastValue = list.get(getSize()-1); //Gets Last Value

                list.remove(getSize()-1); //Removes Last Value

                return lastValue; //Returns Last Value

        }//pop Method

        public void push(E value) {

                list.add(value);


        }//Push Method


}//GenericStack Class
```