```java
import java.util.Scanner;
import java.util.ArrayList;
import java.util.Iterator;

import java.io.File;
import java.io.IOException;

/*
 * Name:         Luke O'Brien
 * Class:        Data Structure and Algorithms
 * Section:      002
 * Assignment:   Assignmnet-5
 * Due Date:     26 Feb 2020
 *
 * Desc:
 * The program takes the four files into a Scanner then parses the data.
 * inputing it all into the "GenericStack" Data structure.
 * The program then prints out the individual stacks before merging them
 * then reversing them. After they are reversed, the final product is
 * it then printed out.
 *
 * after both types of stacks have gone through that process, the final
 * stacks for each type is printed out in side by side format using the
 * method printTwoStacks
 *
 * Added methods to make my life easier:
 * printHeader(String);
 * iterator(); //Under GenericStack in order to use iterator in the program
 *
 * ISSUES:
 * I could not for the life of me, sort the last two digits in the while
 * loop inside mergeStacks(). I understand it's because of where my next()
 * is, but if I were to move it, the rest of the method would break.
 */

public class OBrienLukeAssignment5
{
    public static void main(String[] args) throws IOException
    {
        Scanner intFileOne = new Scanner(new File("numbers1.txt"));
        Scanner intFileTwo = new Scanner(new File("numbers2.txt"));

        Scanner strFileOne = new Scanner(new File("mountains1.txt"));
        Scanner strFileTwo = new Scanner(new File("mountains2.txt"));

        GenericStack<Integer> intStackOne = new GenericStack<>();
        GenericStack<Integer> intStackTwo = new GenericStack<>();
        GenericStack<Integer> intFinStack = new GenericStack<>();

        GenericStack<String> strStackOne = new GenericStack<>();
        GenericStack<String> strStackTwo = new GenericStack<>();
        GenericStack<String> strFinStack = new GenericStack<>();

        GenericStack<Integer> intPrintStack = new GenericStack<>();
        GenericStack<String> strPrintStack = new GenericStack<>();


        while(intFileOne.hasNextInt()) {
            intStackOne.push(intFileOne.nextInt());
        }
        while(intFileTwo.hasNextInt()) {
            intStackTwo.push(intFileTwo.nextInt());
        }
        while(strFileOne.hasNextLine()) {
            strStackOne.push(strFileOne.nextLine());
        }
        while(strFileTwo.hasNextLine()) {
            strStackTwo.push(strFileTwo.nextLine());
        }

        printHeader("Number Stack 1, Filled with Integers from 'number1.txt'");
        printStack(intStackOne);
        printHeader("Number Stack 2, Filled with Integers from 'number2.txt'");
        printStack(intStackTwo);
        mergeStacks(intStackOne, intStackTwo, intFinStack);
        reverseStack(intFinStack,intPrintStack);
        printHeader("Merged numbers stack");
        printStack(intPrintStack);

        printHeader("String Stack 1, Filled with Integers from 'montains1.txt'");
        printStack(strStackOne);
        printHeader("String Stack 2, Filled with Integers from 'montains2.txt'");
        printStack(strStackTwo);
        mergeStacks(strStackOne,strStackTwo, strFinStack);
        reverseStack(strFinStack, strPrintStack);
        printHeader("Merged String stack");
        printStack(strPrintStack);

        printTwoStacks(intPrintStack, strPrintStack);

        intFileOne.close();
        intFileTwo.close();
        strFileOne.close();
        strFileTwo.close();
    }

    public static<T> void printStack(GenericStack<T> stack)
    {
        Iterator<T> iterStack = stack.iterator();

        while(iterStack.hasNext()) {
            System.out.println(iterStack.next());
        }
    }
    public static<T, F> void printTwoStacks(GenericStack<T> stackOne, GenericStack<F>
stackTwo)
    {
        int stackOneSize = stackOne.size();
        int stackTwoSize = stackTwo.size();

        Iterator<T> iterOne = stackOne.iterator();
        Iterator<F> iterTwo = stackTwo.iterator();

        System.out.println("---------------------------------------------");
        System.out.printf("%-15s%s\n", "Integer", "String");
        System.out.println("---------------------------------------------");

        if(stackOneSize >= stackTwoSize) {
            while(iterTwo.hasNext()) {
                System.out.printf("%-15s%s\n", iterOne.next(), iterTwo.next());
            }
            while(iterOne.hasNext()) {
                System.out.printf("%-15s%s\n", iterOne.next(), "-----");
            }
        }
        else {
            while(iterOne.hasNext()) {
                System.out.printf("%-15s%s\n", iterOne.next(), iterTwo.next());
            }
            while(iterTwo.hasNext()) {
                System.out.printf("%-15s%s\n", "-----", iterTwo.next());
            }
        }

    }

    public static<T extends Comparable<T>> void mergeStacks(GenericStack<T> stackOne, GenericStack<T> stackTwo, GenericStack<T> stackFinal)
    {
        Iterator<T> iterOne = stackOne.iterator();
        Iterator<T> iterTwo = stackTwo.iterator();

        T one = iterOne.next();
        T two = iterTwo.next();


        while(iterOne.hasNext() && iterTwo.hasNext()) {
            if(one.toString().compareTo(two.toString()) >= two.toString().compareTo(one.toString())) {
                stackFinal.push(one);
                one = iterOne.next();
            }
            else if(one.toString().compareTo(two.toString()) < two.toString().compareTo(one.toString())) {
                stackFinal.push(two);
                two = iterTwo.next();
            }
        }
        if(one.toString().compareTo(two.toString()) >= two.toString().compareTo(one.toString())) {
            stackFinal.push(one);
        }
        else if(one.toString().compareTo(two.toString()) < two.toString().compareTo(one.toString())){
            stackFinal.push(two);
        }
        while(iterTwo.hasNext()) {
            stackFinal.push(iterTwo.next());
        }
        while(iterOne.hasNext()) {
            stackFinal.push(iterOne.next());
        }
    }
    public static<T> void reverseStack(GenericStack<T> inputStack, GenericStack<T> outputStack)
    {
        while(!inputStack.isEmpty()) {
            outputStack.push(inputStack.pop());
        }
    }

    public static void printHeader(String str) {
        System.out.println("\n\n*********************************************************");
        System.out.printf("%s\n", str);
        System.out.println("*********************************************************");
    }
}

class GenericStack<E> implements Iterable<E>
{
    private ArrayList<E> genStack;

    GenericStack(){
        genStack = new ArrayList<>();
    }

    public void push(E item) {
        genStack.add(item);
    }

    public E peek() {
        return genStack.get(genStack.size()-1);
    }

    public E pop() {
        E temp = genStack.get(genStack.size()-1);
        genStack.remove(genStack.size()-1);
        return temp;
    }

    public int size() {
        return genStack.size();
    }

    public boolean isEmpty() {
        if(genStack.size() == 0) {
            return true;
        }
        else {
            return false;
        }
    }

    public Iterator<E> iterator() {
        return genStack.iterator();
    }
}

```