**CS 1450 Data Structures and Algorithms – Spring 2020**

**Assignment #5**

Due Date: **Feb 26, 2020** at 1:40pm (MW class), **Feb 27, 2020** at 1:40pm (TR class)

Purpose: Learn to write a generic class, generic methods, plus create and use stacks.

Effort: Individual

Points: **100**

Deliverables: Upload a **.zip** file with **ONLY** your source code (.java file) to Canvas.

**Hand in a hardcopy version of the code and your design notes at beginning of class.**

**Assignment Description**

This assignment provides the opportunity to implement and use a generic class and generic methods. The goal: **combine two sorted stacks** (min value on top) into **one sorted stack** (min value on top). The code must work for **any** 2 sorted stacks. The picture shows 2 stacks of integers being merged.

**Stack1 Stack2 MergedStack**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 7 |  | 5 |  | 5 |
| 12 |  | 18 |  | 7 |
| 32 |  | 24 |  | 12 |
| 48 |  | 30 |  | 18 |
|  |  | 42 |  | 24 |
|  |  | 47 |  | 30 |
|  |  |  |  | 32 |
|  |  |  |  | 42 |
|  |  |  |  | 47 |
|  |  |  |  | 48 |
|  |  |  |  |  |

In this assignment, first work from the **developer perspective** and create a generic class for the stack. Once the generic stack is implemented, work from the **user perspective** (code inside main) using the generic stack methods to push objects of type **Integer** onto 2 stacks then objects of type **String.** Finally, write the **generic methods** to merge the two stacks, reverse a stack, and print the objects in a stack.

**Specifications**

1. Create a Java class called **LastnameFirstnameAssignment5**
2. Follow "CS1450 Programming Assignments Policy"
3. Write a test program (i.e. main) that performs the following:
   1. Create 2 **GenericStack** objects of type **Integer** (you will repeat steps 3a-3f for **String**)
      1. See definition of **GenericStack** class below for the class you need to create
   2. Fill the 2 **GenericStacks** with values read from the files:
      1. Read values in **numbers1.txt**
      2. Store values in 1st **GenericStack** – numStack1
      3. Read values in **numbers2.txt**
      4. Store values in 2nd **GenericStack** – numStack2
   3. Print the 2 stacks (write a generic ***printStack*** method)
      1. Print each stack proving they contain the file values (min on top)
      2. Iterate through the stack, popping and printing values.
      3. Write the code so that the stack is in its **original state** after ***printStack*** completes. For example, if the stack contains the following values before calling printStack, then it must contain these values in the same order after the call:

|  |  |
| --- | --- |
| 7 | State of stack before and after  call to ***printStack***must be same |
| 12 |
| 32 |
| 48 |  |

* + 1. The method signature is left as an exercise but it should have **only one** stack formal parameter. This means, you need to call it once for each stack.
  1. Merge the 2 stacks (write a generic ***mergeStacks*** method)
     1. First create 1 **GenericStack** object of type **Integer** (this is the **mergedStack**)
     2. Call ***mergeStacks*** to combine numtack1 & numStack2 into **mergedStack**.
     3. Note: merging causes the largest value to be on top in the mergedStack. The reversal is done by another method because methods should perform 1 task.
     4. Make sure you understand the syntax of this method call, that is, what is <E extends Comparable<E>>> for, what is the return type, etc.

**public** **static** <E **extends** Comparable<E>> **void** mergeStacks

(GenericStack<E> stack1,

GenericStack<E> stack2,

GenericStack<E> mergedStack)

* 1. Reverse a stack (write a generic ***reverseStack*** method)
     1. First create 1 **GenericStack** object of type **Integer** (this is the **finalMergedStack**)
     2. Call ***reverseStack*** to reverse values in mergedStack so smallest value is on top

**public** **static** <E> **void** reverseStack (GenericStack<E> stack,

GenericStack<E> finalMergedStack)

* 1. Print the final merged stack
     1. Call generic ***printStack*** method described in step c to print **finalMergedStack**.
  2. **REPEAT STEPS 3a – 3f** for the two **String** files (***mountains1.txt*** and ***mountains2.txt***)
  3. Print 2 merged stacks side by side (write a generic ***printTwoStacks*** method)
     1. Call generic ***printTwoStacks*** to print the contents of two stacks side by side.
     2. The method ***printTwoStacks*** has two generic parameters, E and F, since the stacks are of different types.

**public** **static** <E, F> **void** printTwoStacks(GenericStack<E> stack1,

GenericStack<F> stack2)

* + 1. This method may use one temporary stack of type GenericStack<E> and one temporary stack of type GenericStack<F>. It may also use one temporary variable of each type, E and F.
    2. No other data structures may be used (array, ArrayList, etc.)
    3. The stacks will likely be of different lengths, so when one stack runs out of elements, fill in the blanks in its display with “----“.
    4. The method must leave the stacks in their original state, to prove this, print the top element in each stack after the ***printTwoStacks*** method has returned.
  1. Note:
     1. There is only one ***printStack***, one ***mergeStack,*** one ***reverseStack,*** and one ***printTwoStacks*** method.
     2. These methods belong after main since they belong to **class Assignment5**
     3. **DO NOT** create these methods for Integers and another set of methods for Strings - that defeats the purpose of learning to use generic methods.

1. Test files information:
   1. There are **4 files**: two files with integer values and two files with string values.
   2. The values are stored in reverse sorted order in the files so as you read and push the values onto the stack the stack ends up in proper sorted order (min value on top).
   3. Remember, these files are examples so do NOT ASSUME file size or specific values beyond Integers and Strings.

|  |  |  |  |
| --- | --- | --- | --- |
| **numbers1.txt** | **numbers2.txt** | **mountains1.txt** | **mountains2.txt** |
| 48  32  12  7 | 47  42  30  24  18  5 | Snowmass Mountain  Pikes Peak  Mt. Sneffels  Mt. of the Holy Cross  Mt. Elbert | Mt. Princeton  Mt. Evans  Maroon Peak  Longs Peak |

**Class**

**GenericStack Class**

* Description:
  + A stack is a data structure with last-in, first out behavior.
    - Last item added is first item removed.
    - Visually, it’s like a stack of plates – where plates are added to the top of the stack, and plates are also removed from the top of the stack.
  + To be a generic class it must look like this:

**class** GenericStack<E> **E is a placeholder. When creating a**

**GenericStack indicate the type - Integer, String**

* Private Data Fields
  + **list** – Use an **ArrayList** as the storage container (i.e. structure to hold objects in stack)
    - To implement a stack, we add and remove only from **one end** of the ArrayList.
    - Allow the ArrayList’s **last** occupied location to represent the **top** of the stack.
    - Now the **last element** added to the ArrayList is the **1st element** removed
* Public Methods
  + Constructor: ***public******GenericStack()***
    - Creates an empty stack, that is, creates an empty ArrayList
  + ***getSize***
    - Returns the number of objects currently on the stack
  + ***isEmpty***
    - Indicates if the stack is currently empty
  + ***peek***
    - Returns the object on the top of the stack, does NOT remove that object
  + ***push***
    - Adds an object to the top of the stack (i.e. end of the ArrayList)
  + ***pop***
    - Removes the object on the top of the stack (i.e. end of the ArrayList)

**Tips**

**Tip: Writing a Generic Class**

* When first working with generics, it can be difficult to understand and write generic code.
* Some students find it helpful to use this approach to learn generics:
  + 1st write an **Integer** stack class **without** any generics.
  + Use **Integer** as the type for ArrayList and the type in methods (arguments/return type).
  + Test the class and get the regular **Integer** stack working properly.
  + Convert **Integer** class to a generic class by replacing all **Integer** types with the generic type parameter **E.** Make sure the code still works.
  + Finally, write code for Strings using generic class & methods and everything should work

**Tip: Comparing Values in a Generic Stack**

* If you attempt to compare stack values using the relational operators, you will see this error
  + The operator > is undefined for the argument type(s) E
* The ***mergeStacks*** method is bounded by comparable so using the **compareTo** method solves this error.

**Output**

Output when running against the test files **numbers1.txt, numbers2.txt** and **mountains1.txt**, **mountains2.txt**

Numbers Stack 1 - filled with values from numbers1.txt

-------------------------------------------------------

7

12

32

48

Numbers Stack 2 - filled with values from numbers2.txt

------------------------------------------------------

5

18

24

30

42

47

Merged Stack - lowest value on top

-----------------------------------

5

7

12

18

24

30

32

42

47

48

String Stack 1 - filled with values from mountains1.txt

-------------------------------------------------------

Mt. Elbert

Mt. Of The Holy Cross

Mt. Sneffels

Pikes Peak

Snowmass Mountain

String Stack 2 - filled with values from mountains2.txt

-------------------------------------------------------

Longs Peak

Maroon Peak

Mt. Evans

Mt. Princeton

Merged Stack - lowest value on top

-----------------------------------

Longs Peak

Maroon Peak

Mt. Elbert

Mt. Evans

Mt. Of The Holy Cross

Mt. Princeton

Mt. Sneffels

Pikes Peak

Snowmass Mountain

Printing Merged Stacks side by side - lowest value on top

-------------------------------------------------------------

Integers Strings

-------------------------------------------------------------

5 Longs Peak

7 Maroon Peak

12 Mt. Elbert

18 Mt. Evans

24 Mt. Of The Holy Cross

30 Mt. Princeton

32 Mt. Sneffels

42 Pikes Peak

47 Snowmass Mountain

48 ----

Number stack top: 5

String stack top: Longs Peak