Stack Client

Goal

In this assignment you will complete an application that uses the Abstract Data Type (ADT) stack.

Resources

- Chapter 5: Stacks

In `javadoc` directory

- *`StackInterface.html`*—Interface documentation for the interface `StackInterface`

Java Files

- *`StackInterface.java`*
- *`StackSort.java`*
- *`VectorStack.java`*

Introduction

In computer science, one of the important basic structures is the stack. It is of both theoretical and practical use. In its simplest form it has three operations: push, pop, and empty. Push places a value on the top of the stack. Pop removes the top value from the stack. Empty is a test to determine if the stack has any values in it. Some specifications give a fourth operation called peek (or top). Peek will return the top value on the stack but leaves the number of items unchanged. Strictly speaking, peek is unnecessary because a pop followed by a push will mimic its operation. Before continuing the assignment you should review the material in Chapter 5. In particular, review the documentation of the interface *`StackInterface.java`*.

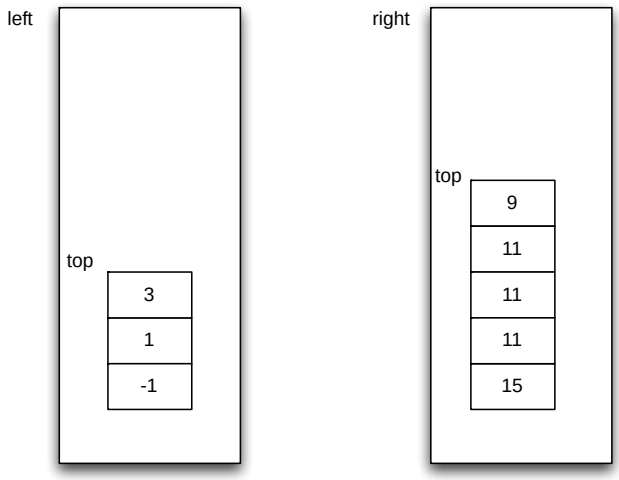
In theoretical computer science, one of the problems of interest is recognizing words in a language. In this context, a language is a set of words that follow some pattern. For example, one language is all the words from the alphabet {0, 1} that have equal numbers of zeros and ones. The word 001011 is in the language, but the word 00111 is not. There are a number of primitive models of computation that have different abilities. One kind of model is a machine called a pushdown automata (PDA). It has a finite control (program) and a single stack that it can use for memory. While fairly powerful, a PDA does have some surprising limits. For example, while a PDA can recognize words of the form 0^n1^n, it cannot recognize $0^n1^n0^n$. Modern computer languages are often recursively defined by a grammar, which can be recognized by a PDA.

The application you will complete implements a sorting algorithm. Sorting is a general problem where given a collection of items, you arrange them in order from smallest to largest. We will restrict ourselves to a collection of integer values in an array. For example, if given the integers 8, 2, 9, 1, 1, 3; their sorted order is 1, 1, 2, 3, 8, 9. You will examine a number of different sorting techniques in Chapters 8 and 9. While it is not obvious, the sort that we will be doing in this assignment is equivalent to the insertion sort from Chapter 8 and has the same performance. As with any of the sorts from Chapter 8, the stack sort should not be used in general applications.

Visualization

Stack Sort

In order to sort values we will use two stacks which will be called the left and right stacks. The values in the stacks will be sorted and the values in the left stack will all be less than or equal to the values in the right stack. The following example illustrates a possible state for our two stacks. Notice that the values in the left stack are sorted so that the smallest value is at the bottom of the stack. The values in the right stact are sorted so that the smallest value is at the top of the stack. If we read the values up the left stack and then down the right stack, we get -1, 1, 3, 9, 11, 11, 11, 15, which is in sorted order.



Suppose that we have a new value that we want to put into our sorted collection. We will want to put it on the top of one of the two stacks, but we may have to first move values around.

No moves required:

Consider adding the value 5 to the example shown above. We do not have to move any values and can place the 5 on the top of either stack and still have a sorted collection.

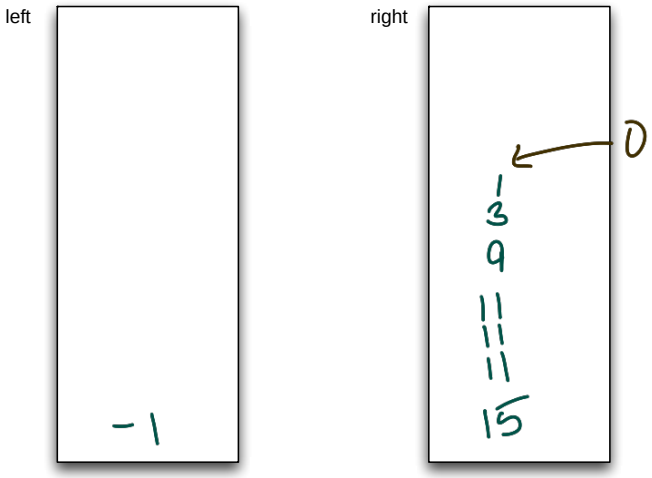
Which values would not require that the contents of the stacks be changed?

$3 \leq x \leq 9$

Moves from left to right required:

Consider adding the value 0 to the example shown above. We must move values from the left stack to the right stack.

How many values must be moved and what is the state of the two stacks before we add the value 0?

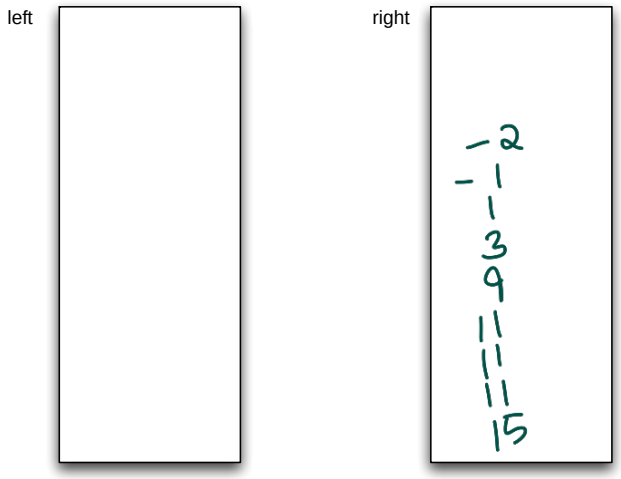


What condition should we use to determine if enough values have been moved?

IF THE VALUE IS BIT LEFT TAIL & RIGHT TAIL

Consider adding the value -2 to the example shown above. Again must move values from the left stack to the right stack.

How many values must be moved and what is the state of the two stacks before we add the value -2?



What condition should we use to determine if enough values have been moved?

SAME AS ABOVE?

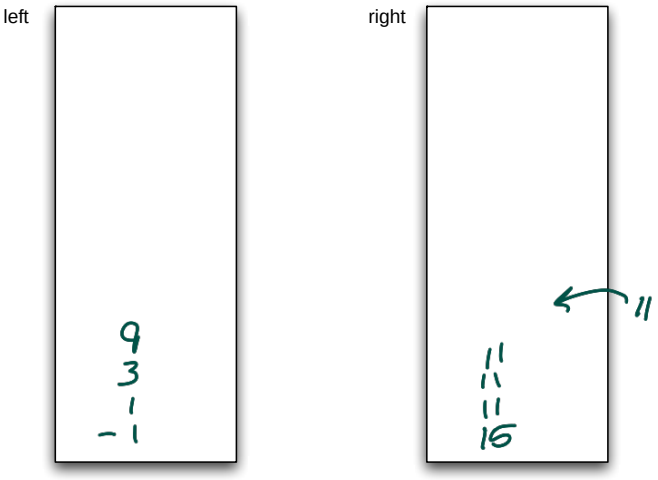
Write code using iteration that will move values from the left to the right stack as required.

```
While (left[left.length-1] ≥ target){
    right[length]=left[length-1];
    left[length-1]=null;
}
```

Moves from right to left required:

Consider adding the value 11 to the example shown above. We must move values from the right stack to the left stack.

How many values must be moved and what is the state of the two stacks before we add the value 11?

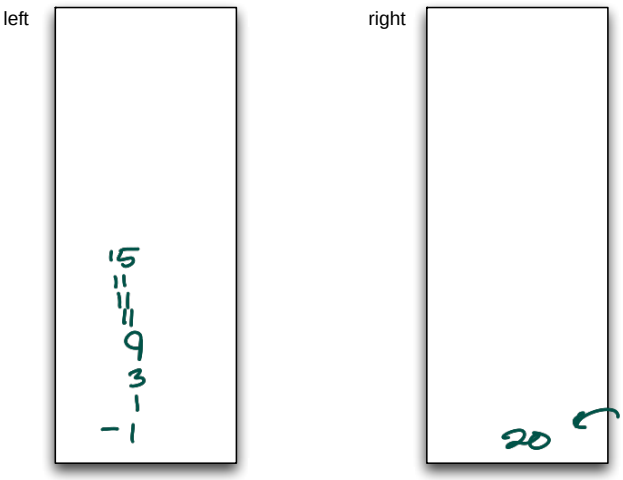


What condition should we use to determine if enough values have been moved?

IF END OF LEFT

Consider adding the value 20 to the example shown above. Again we must move values from the right stack to the left stack.

How many values must be moved and what is the state of the two stacks before we add the value 20?



What condition should we use to determine if enough values have been moved?

SAME AS ABOVE

Write code using iteration that will move values from the right to the left stack as required.

```
While(right[length-1] <= target){
    left[length-1] = right[length-1]
    right[length-1] = null;
}
```

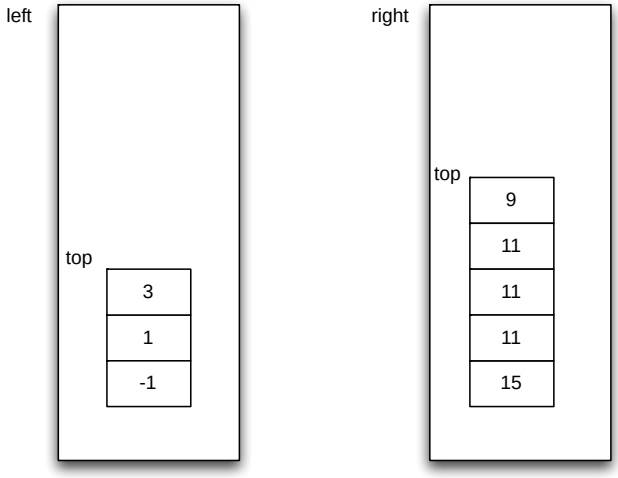
Adding all the values from an array into the two stacks:

We can add the values from the array one at a time to the stacks. Putting together the pieces from the previous questions, write an algorithm for this task.

```
for(int i = 0; i < arr.length; i++){
    boolean adding = true;
    while (adding) {
        if (look above) { move elements }
        else if (look above) { move elements other way }
        else { add element; adding = false; }
    }
```

Putting the values into a new array:

For our particular sorting algorithm, we are going to create a second array with the values from the original array in sorted order. Therefore, the final task we need to do before we return is to put the values into the `result` array. Consider again our example.



Suppose we pop the values off of the left stack one at a time. What order do we get?

3, 1, -1

Suppose we pop the values off of the right stack one at a time. What order do we get?

9, 11, 11, 11, 15

This suggests that if we move the values from the left stack to the right stack, we can then directly pop them off of the right stack into the `result` array. Write an algorithm that accomplishes this task.

```
While ( left.length != 0 ) {
    right[length] = left[left.length-1];
}
for (int i=0; i<right.length; i++) {
    result[i] = right[right.length-1-i]
}
return result;
```

Directed Work

Stack Sort

Pieces of the `StackSort` class already exist and are in *`StackSort.java`*. Take a look at that code now if you have not done so already. Also before you start, make sure you are familiar with the methods available to you in the `VectorStack` class (check *`StackInterface.html`*).

Step 1. Compile the classes `StackSort` and `VectorStack`. Run the `main` method in `StackSort`.

Checkpoint: If all has gone well, the program will run. It will create arrays of various sizes and print out the result of `StackSort` method. At the end, the program will ask you to enter an integer value. It will use the value to create and then sort an array of that size. Enter any value. The sorted arrays as reported by the program should all be empty. Our first goal is to get values into a stack and then move them to the result array.

Step 2. Create a new `VectorStack<Integer>` and assign it to `lowerValues`.

Step 3. Create a new `VectorStack<Integer>` and assign it to `upperValues`.

Step 4. Using a loop, scan over the values in the argument array `data` and push them onto the `upperValues` stack.

Step 5. Using a loop, pop all the values from the `upperValues` stack and place them into the array `result`.

Checkpoint: Compile and run the program. Again it should run and ask for a size. Any value will do. This time, you should see results for each of the calls to the StackSort method. The order that values are popped off the stack should be in the reverse order that they were put on the stack. If all has gone well, you should see the values in reverse order in the results array. We will now complete the StackSort method

Step 6. Inside the loop that scans over the `data` array, we need to move the data between the two stacks before we push the value. Refer to the visualization exercise to complete the body of the loop.

Step 7. Before the loop that pops the data values from the `upperValues` stack, we need to move any data values from the `lowerValues` stack. Refer to the visualization exercise to implement this loop.

Checkpoint: Compile and run the program. The output of the StackSort method should be the original arrays in sorted order. If not, debug until the results are correct.