CS2001 Practical 2 Report

Matriculation ID: 220006323
2023-10-30

Overview

In this practice I was asked to implement and test the given interfaces of IStack IDoubleStack and IQueue by making classes. I have fully met the specification requirements by implementing all the classes needed to meet the interface restrictions and following through by fully testing each aspect using JUnit tests.

Design

For the design I was restricted by the interfaces, the IDoubleStack would use two getters for the two stacks. The IStack would have a push method to add objects to a stack. The pop method to retrieve the element last added and remove it from the stack. The top method would do the same as the pop method without removing it. I would also need to have methods to clear a stack, find its size and check whether a stack is empty.
The IQueue would use the double stack interface to implement a Queue. I would add elements to the queue using the enqueue method and also pop them using the dequeue method. I would also have methods for size, clear and checking the queue is empty.

Implementation

As per the requirements of the specification I created a Stack.java file that through its constructor would take in an object array and boolean flag. The Object array is the shared array created in the double stack. The flag allows the code to identify if the stack is the first or second stack.

If the element to add to the stack is null i throw a null pointer exception. Depending on if the stack is the first or second stack. If the stack is the first one then I check whether the size exceeds more than half the shared array size. I use the attribute top that is assigned the value -1 initially. If the stack is the second one then it checks that the top2 variable is greater than half the shared array size. If either of these conditionals succeed then a stack overflow exception is thrown.

For the first stack if an exception is not thrown then I add the element to the array based on the top value. I use ++top to increment and use it.

For the second stack if an exception is not thrown then i also add to the shared array but from --top2 index. Top2 is assigned the value of the stack size on initialization. This populates the array backwards as advised in the hints of the specification.

The Pop method
If it is the first stack then it checks whether the top value is greater or equal to 0. If this isn't the case then a StackEmptyException is thrown. Otherwise it retrieves the element at the top index of the array for that stack. It then assigns that array element to be null and decrements the top attribute.

For the second stack I check that the top2 variable is less than the total size of the array. If this isn’t the case then a StackEmptyException is thrown. Otherwise The same is done as before but top2 attribute is incremented instead.

Both return the element at the top of the array whilst deleting it from the array.

The top method.
This does the same as the pop index with the same code apart from excluding assigning the array element to be null and incrementing or decrementing the top2 and top value respectively.

The size method
For the first stack I retrieve the top attribute and add one to it.
For the second stack I find the difference between array size and top2 value.

isEmpty method
I check if the size of the stack is 0
If it is 0 then I return true and if not I return false.

clear method
For the first stack I loop from the top value to decrementing till 0 and assign each of the elements as null.
I then assign top attribute to -1
For the second stack I loop from the top2 value upto the highest index of the array and assign each element to be null.
I assign top2 back to equal the size of the array.

DoubleStack.java file
In the constructor I take in the maxSize. This is used to create an object array called shareArr with that size.
I use this array when initializing two stacks called first and second as well as giving the first stack the true flag and the second stack the false flag.
I assign the first stack to the 0 index of the attribute stack array called arr and the second stack to index 1.
The getFirstStack method returns index 0 of the arr stack array
The getSecondStack method returns the element in index 1 of the arr stack array.

DoubleStackQueue.java file
In the constructor I create I initialise a DoubleStack object called init.
I use this to assign the first stack to the input attribute and the second stack to the output attribute.

Enqueue method
When I add an element to the queue I first check that the size of the input stack is not equal to half of the shared array size. If this is the case then a QueueFullException is thrown.

I then use the push method to add the element to the input stack.

Dequeue method
I check if the input array is empty. If this is true then I will throw a QueueEmptyException.
Otherwise I check that the output stack is empty. If this is the case then I use a for loop to push and pop each element in the input stack into the output stack. I then pop the output stack and return that value.

Size method
I add the size of the input and output stack and return it to provide the size of the queue.

IsEmpty Method
I check whether both the input and output stacks are empty using the isEmpty method for each.If this is the case then I return true otherwise false.

Clear method
I clear both the input and output stack using the clear method for each.

Testing

TestArrayDoubleStack.java file
factoryReturnsNonNullDoubleStackObject
Asserts that the double stack object is non null

pushpop()
Pushes (3) to the first stack and pops it and asserts that the popped value is 3.

pushpopsecond()
Does the same as pushpop() but for the second stack.

pushpopsecond2()
Does the same as pushpopsecond() but also has elements pushed to the first stack.

pushpopfirst2()
Does the reverse of pushpopsecond2() for the first stack instead

pushpopmultiple()
I push multiple elements (3,7) to the first stack. I then pop and check that the element that is popped is the last added(7)

pushpopmultiplesecond()
Same as before but with second stack instead

pushpopOverflowSecond()
Adds 6 elements to the second stack with array size being 10
Checks that stackoverflowexception is thrown using assert True and thrown boolean

pushpopOverflowFirst()
Does the same for the first stack

pushpopOverFlowBoth()
Does the same but only adds 5 per stack and therefore checks that a exception is not thrown using assert False.

pushpopOverFlowFirstOdd()
The array has size 11(odd) I do the same as pushpopOverFlowFirst adding 6 elements and check that a exception is thrown in the same way.

pushpopNoOverFlowBothOdd
I add 5 elements to stack 1 and 6 to stack 2 with the array size of 11 and check that no exception is thrown.

pushpopOverFlowBothOdd
Same as pushpopOverFlowBoth but with 11 size and induces and exception by having 6 elements pushed to the first stack and only 5 in the second. I made it so that if the size is odd then the second stack takes in one more element.

popEmpty()
I pop the first stack while it is empty.
Check that a StackEmptyException is thrown using the same method

popEmptySecond()
Same as popEmpty but using the second stack

popEmptySecondFirstNot()
Checks that behavior of isEmpty is same regardless if the other stack is not empty

popEmptyFirstSecondNot()
Reverse of the test before it

pushpushpopop()
Double push and then double pop
assertEquals that each pop is correct. Pushing 3 then 7 and popping 7 then 3

pushpushpopopSecond()
Does the same but with the second stack

size()
I add two elements and then make sure the size of the first stack is 2

size2()
I add two elements to the first stack and 1 to the second and check that the sizes match accordingly.

isEmpty()
Checks that the first and second stack are empty after initialization

clear()
Push two elements to the first stack and one to the second. I make sure isEmpty for the first and second is false
I then clear the first stack and make sure isEmpty is now true
I then clear the second and do the same.

pushpushpopopsize()
Do the same as pushpushpopop but also check that size of stack changes accordingly to each pop

pushpushtoptopsize()
I do the same as before but using the top value. This makes sure size doesn't change and u keep getting the same element using the top method

pushpushtoptopsizeSecond()
Same as before but for the second stack

pushpopString()
Same as pushpop but using the string “Test” instead of int 3

pushpopNull()
Same as pushpop but pushing null
Makes sure a nullpointerexception is thrown.

clear2()
Same as clear but with a full two stacks.

TestDoubleStackQueue.java file
factoryReturnsNonNullDoubleStackQueue
Asserts that the doublestackqueue object is nonull

enqueuedequeue()
I queue the element 3 and then assert that 3 is the value received from dequeuing it. I also make sure that the queue size is 0 after dequeueing.

dequeueEmpty()
Induce a QueueEmptyException by a dequeuing an empty queue.
Using the same method as before to check a thrown exception.

enqueueOverfllow()
Adding 6 elements to the queue without dequeueing. This causes Queue full exceptiona and is checked using the same method.

enqueueOverFlowODD()
Same as before but with size of 11.
The implementation means that 6 elements will still overflow it.
The output queue is the only that can take 6 in this scenario.

enqueueenqueuedequeuedequeue()
Checks that with two enqueues and two dequeues that I receive the first enqueued element which is 3 and then 7 when dequeueing.

EnqueueDequeueEnqueueDequeue()
I do the same as before but in the order outlined in the name.

size()
I add two elements and make sure that the size is 2
I then dequeue an element and see it drop to 1.

size2()
I enqueue 3 elements and dequeue one and then enqueue one. I check that the size is 3.

isEmpty()
Check that queue isEmpty returns true after initialisation

isEmptyFalse()
Is false after I add element 3 to the queue

isEmptyFalseDequeue()
I add two elements and then dequeue one. I check that isEmpty is false since there is one element remaining

clear()
I add an element and clear it. I then check that the queue is empty

clearDequeue()
I add two elements and dequeue one then clear the queue.
I check that the queue is empty

enqueueNull()
I enqueue an element value null. I check that a nullpointerexception is thrown.

Evaluation

In this submission I have fully met the practical specifications of making classes to implement the interfaces provided.

My classes implement the required methods in a way that matches the described functionality from the comments.
I created many comprehensive tests for each method and class using JUnit.

I was able to successfully implement the double stack in which one array is shared among two stacks. Each stack can use the interface methods and functionally changes based on if the stack is first or second.

The DoubleStackQueue uses the double stack to implement a working queue system using an Input and Output stack.

Conclusion

I have fully met the specification. In this practical I found most difficult was conceptualizing the shared array between two stack systems and how I would implement this in my Stack.java file.

Given more time I would think of more tests and edge cases that elude my awareness right now.