HP35

Abdinasir Abdi

September 2024

1 Introduction

This report concludes the second assignment in course ID1021 in the fall semester of 2024. In this assignment, a calculator will be implemented using reverse Polish notation. The goal of this is to get a better understanding of what a stack is and how it can be used. Two versions of a stack will be implemented, a static and dynamic stack. When creating the stack it has to be done from scratch, Java libraries such as ArrayList are not allowed.

2 Static Stack

A static stack has a predetermined fixed size, is implemented with arrays, and is simple to construct and use. They are very efficient and usually used for small, predictable workloads. The size of the array can not be changed unless explicitly done. The time complexity is constant $O(1)$ for "push" and "pop" operations, except for when the stack is full.

```
int[] stack;
    int top;

    public StaticStack(int size) {

        stack = new int[size]; // Allocate memory for stack
        top = -1; // Empty stack
    }
```

The first thing made was initializing and allocating memory for the stack and initializing a pointer index to keep track of the top element on the stack.

```
     public void push(int val) {
        if (top == stack.length - 1) {
            System.out.println("Stack is full");
        } else
            stack[++top] = val;

    }

    public int pop() {
        if (top == -1) {
            System.out.println("Stacken is empty");
            return -1;
        } else
            return stack[top--]; .

    }
```

The next step involved constructing the methods for adding and taking elements from the stack, i.e. ”push” and ”pop”. Both methods are simple to construct because they follow the principle of last in first out(LIFO). This means that when an element is added it's on the top position of the stack and therefore also the first element out. The push method simply adds any given element on top of the stack and increments the pointer index(top). The pop method does the opposite, first returning the top element and decreasing the pointer index.

In both methods the situation where an operation is attempted on an empty stack, is addressed. In push it checks if the pointer index is at the last position in the stack, if true it means that the stack is full and no element will be added. In pop it checks if the pointer index is at the position before the first position, if true this would mean that the stack is empty. With this, a simple stack was created where elements could be added and removed.

3 Dynamic Stack

A dynamic stack follows the same principles as a static stack, mainly the key difference being that with dynamic the size of the stack is flexible. Dynamic is usually implemented with a linked list or dynamic arrays. For this assignment dynamic arrays will be implemented. Two new methods have been added. A method that doubles the size of the array and a method that shrinks the array when certain requirements are met.

```
    public void double_size() {
    size *= 2;
    int[] nyaStack = new int[size];
```

```
     for (int i = 0; i < top; i++) {
        nyaStack[i] = stack[i];
    }
    stack = nyaStack;
    System.out.println("Resized stack to size: " + size);

 }
```

The method for doubling is quite straightforward, simply create a new array with twice the size and then copy all the elements into that new array.

```
    public void shrink_stack(){

        // Ensure the stack doesn't shrink below the minimum size
        int newSize = Math.max(size / 2, Min_Size);
        if (newSize != size) {
            int[] nyaStack = new int[newSize];

            for (int i = 0; i < top; i++) {
                nyaStack[i] = stack[i];
            }
            stack = nyaStack;
            size = newSize;
            System.out.println("Shrunk stack to size: " + size);
        }

    }
```

The method for making the array smaller over time is somewhat more complex. A new array is created and by using Math.max(a,b) which returns the larger value of a and b, the array will shrink by half, but not to less than the specified minimal size. Continuing by copying all the elements into the new stack and also updating the size of the stack.

With the establishment of these methods, the pop methods get new variables. Variables that will function as limits/parameters for the stack.

```
    int poppedValue = stack[--top];
            popCount++;

if (top > 0 && top <= size / 4 && popCount
>= SHRINK_THRESHOLD && size > Min_Size) {
shrink_stack();
popCount = 0;  // Reset pop counter after shrinking
}

          return poppedValue;
```

There is now a counter that keeps track of the how many elements that are popped. A shrink threshold is predetermined and the counter for the amount of pops functions as the indicator for when the threshold is reached. With this, the array gets continuously smaller as we pop more elements and doubles in size when the stack is full.

4 Calculator

The final part of the assignment was about making a calculator using reverse Polish notation, RPN. RPN is a mathematical notation that eliminates the need for parenthesis to define operation order. Operators follow their operands and the position in which they are pushed into the stack determines the calculation sequence.

Using the created methods for the functionality of the stack and the provided skeleton code for the calculator, different cases were implemented. For this calculator, only the operand addition, subtraction, and multiplication were needed.

```
    case "+":
                     int a = stack.pop();
                     int b = stack.pop();
                     int result = (a + b);
                     stack.push(result);
                     break;
```

The same logic was used for subtraction and multiplication. With these, a semi-complete calculator was made and the goal was to try solving the following:

$423 * 4 + 4 * +2-$

The RPN process works in a left-to-right manner, meaning 4 would be the first element pushed into the stack. Thereafter numbers 2, 3, and 3 would be on top of the stack. The two highest elements are multiplied and pushed back on top of the stack. In the next operation, 4 is added to the stack and below there is a visual representation for the rest of the notations. The array follows a right-to-left manner meaning that the element on the rightmost position is the top element on the stack.

$$[4, 10, 4] \tag{1}$$
$$[4, 40] \tag{2}$$
$$[44, 2] \tag{3}$$
$$[42] \tag{4}$$

When written to the calculator 42 is the output result. This is correct per the expected answer.